The University of Southampton

Academic Year 2014/2015

Faculty of Social and Human Sciences

Mathematics

MSc Dissertation

Heuristics algorithms for vector bin packing problem

Hana Elgaramali

A dissertation submitted in partial fulfilment of the MSc in Operational Research

This project is entirely the original work of Hana Elgaramali. Where material is obtained from published or unpublished works, this has been fully acknowledged by citation in the main text and inclusion in the list of references.

Word Count: 11614 words

Acknowledgement

I have the pleasure of expressing my appreciation and gratitude to a number of people who encourage, support and assist me to finish my dissertation.

First of all, I would like to thank my supervisor, Dr. Antonio for his excellent guidance, instructions and recommendations that all help me to finish the dissertation effectively.

I would like also to thank my parents, who are the basis of my reaching this stage. So, I am extremely grateful to all of the things that they gave me, whether moral or financial support.

Finally, I cannot find words to express my gratitude to my dear husband "Dia" and my son "Fadel" for their wonderful position. I am truly indebted for my husband for his encouragement and moral support throughout this year to finish my Masters. As well for my son, who spent long hours away from him to allow me focus. I am deeply sorry for that time.

Contents

List of Tables

Abstract

In this research, we consider the variable size vector bin packing problem in the case when the dimension $(d = 2)$. This problem is a generalization of the vector bin packing problem where the bins have variable sizes (in our case we have two sizes) and the objective is to pack a set of items into a minimum number of bins. We propose five different strategies for solving the variable size vector bin packing problem, these strategies are based on first fit (FF) algorithm. We perform a computational experiment on two randomly generated sets of instances in order to analyse the empirical performance of these strategies. Each set of items has a fifteen bin types and runs with small number of items up to 150 items and with large number of items up to 3000 items. These proposed algorithms were run twice, in the first case there were an equal number of items in each item type, while in the second case the demand of each type of items is random. Our numerical results show that the algorithms in strategy 5 (algorithm 9 and algorithm 10) which rely on the average size and the weighted average size are considered as the most effective methods to solve the variable size vector bin packing problem since their performance is superior to other strategies.

1. Introduction

Cutting and packing problems $(C\&P)$ is an active field of studies during the past decades. Also Its significance lies in being relevant to the industrial sector and services as well.

1.1. Packing problems

In order to distinguish between the cutting and packing problems $(C\&P)$, Dyckhoff (1990) propose a typology in categorizing the cutting and packing problems, these four criteria are dimensionality, the shape of the assignment, the types of assortments and the availability of the objects. The availability criterion differentiates between bin packing problems and cutting stock problems. In the bin packing problems, there are a little number of small objects while in the cutting stock problems, the small objects are many (cited in Alves and Valério de Carvalho, 2007). However, the packing problems includes a wide variety of problems, Dyckho and Finke (1992) differentiate these problems in terms of items (size and shape) and bins (form and capacity) as well (cited in Fleszar and Hindi, 2002).

1.2. Scope

The variable size vector bin packing problem $(VSVBP)$ is considered in this report due to the importance of multidimensionality in the recent applications, in which it lies in enabling the items to carry several incomparable attributes. For instance of these incomparable attributes, the requirements of the memory and the requirements of the bandwidth in the environment of computation (Rao et al., 2010). Also, minimizing the number of bins used leads to having the (near-) optimal solution which is always desirable, even when obtained high quality solutions. Korf (2002) states the main four reasons behind this orientation, firstly, the character of some of the applications may be sometimes require the existence of optimal solutions. Especially, when looking for a minimum number of bins, even a one more extra bin is comparatively expensive. Secondly, the ability of identifying the optimal solutions could be consider as an accurate measurement in determining the quality of approximate solutions. For example, it is possible to compare the first fit decreasing (FFD) and the best fit decreasing (BFD) solutions because we could compute their optimal solutions. In addition to that, finding optimal solutions through anytime algorithm is beneficial for devising better solutions with respect to running time than those obtained by BFD or FFD algorithms. Indeed, this is important in practice. Finally, optimal bin packing is a challenging computational problem which may be result in better perception that could probably be applied to other problems.

1.3. Outline

The remaining parts are organized as follows. In Section 2, an overview of literature about the bin packing problem and its variants, also the main techniques that is used in solving these problems will be reviewed. Section 3 includes the formulation of the variable size vector bin packing problem, Section 4 shows different strategies that based on first fit (FF) algorithm for solving the variable size vector bin packing problem. In Section 5, results and analysis of those methods will be compared. Finally, conclusion will be in the Section 6.

2. Literature Review

2.1. Bin packing problem (BPP)

2.1.1. Problem definition

Bin packing problems ($BPPs$) are represented as a one of the challengeable combinatorial optimization problems (cited in Haouari and Serairi, 2009). These problems appear as a principal problem or as an important subproblem in several industrial applications (Camacho, Terashima-Marin, Ochoa, and Conant-Pablos, 2013; Fleszar and Charalambous, 2011; Fleszar, 2012 cited in Dokeroglu and Cosar, 2014).

The classical bin packing problem is defined as follows. We are given a set of items and infinite number of bins in which each item has a specific size and each bin has the same capacity. The goal is to pack all of the items into a minimum number of bins while ensuring that the total sizes of all items loaded into a bin does not exceed the bin's capacity.

The bin packing problems could be categorized according to the bin size as single or multiple bin size.

Firstly, introducing the single bin packing problem which could be explained through the one-dimensional bin packing problem. Since this problem is consist of a number of items with given weights and bins of identical size and the goal is to place these items in the minimum number of bins in which it will fit without violating the capacity constraints, which means that the total capacity of the packed items in the bin should not exceed the capacity of the bin (cf. Martello and Toth,1990; Scholl et al., 1997; Schwerin and Wascher, 1997). There are also other names calling for this kind of problem such as Vehicle Loading Problem (cf. Golden,1976, p. 266) and Binary Cutting Stock Problem (cf. Vance et al., 1994). Babel et al. (2004) study another type of this problem named the k-Item bin packing problem, in which for each bin, is allocated no more than k items.

Moving to the two-dimensional (Orthogonal) bin packing problem, where in this problem the purpose is to pack a collection of various rectangles into a minimum number of rectangular bins. However, this type of problem is also mentioned as the two-dimensional finite bin packing problem to know the difference between it and the two-dimensional strip packing problem, where the bins in this one have an infinite size in one dimension (Lodi et al., 1999, 2002b, p. 379; Lodi et al., 2002a, p. 242; Martello and Vigo, 1998). George et al. (1995, p. 693) indicate the cylindrical bin packing problem that is a two-dimensional circular single bin sized bin packing problem where the items are circles and the bins are rectangles. In practice, this type of the problem is appearing in the logistics issues.

Regarding the three-dimensional (Orthogonal) bin packing problem, in this problem is supposed that the items are rectangular boxes and the bins are rectangular with the same capacity (cf. Lodi et al., 2002c). Miyazawa and Wakabayashi (2003) describe a particular case of the three dimensional rectangular bin packing problem, where that all of items and bins are cubes and it has been named the cube packing problem.

Moving to the multiple bin sized bin packing problems, Chu and La, (2001) and Kos and Duhovnik, (2002) consider the one-dimensional variable sized bin packing problem which is a generalization of the classical one-dimensional bin packing problem where a number of bin types are added and each type of these bins has its own cost and size. Also, the number of available bins per bin type is infinite. The aim is to minimize the total costs of the used bins during packing all of the items into bins (cf.Kang and Park, 2003). As well as a specific case "rectangular case" in a two-dimensional is studied by Tarasova et al. (1997) (Cited in Wäscher, Haußner and Schumann, 2007).

2.1.2. Usage BPP in real-world applications

The bin packing problems arises in the context of many real-world applications for example in cutting, packaging, planning of telecommunication, transportation, production, and supply-chain systems.

The two and three dimensional bin packing problems are usually appeared in the manufacture as well. For example, in all of construction, clothing, glass, plastic, or metal industries, the aim is to use the minimum number of sheets of these materials (Dahmani, Clautiaux, Krichen, & Talbi, 2013). In the same way, in the case of designing the page' layout of a newspaper, where the pages have fixed dimensions and it is required to order the articles on them. In the shipping and transportation industries, the minimum number of rectangular bins is required when loading bundles of the same heights (cited in Dokeroglu and Cosar, 2014).

2.1.3. Related work

The first fit decreasing (FFD) algorithm (Eilon and Christofides 1971, Johnson et al. 1974) and the best fit decreasing (BFD) algorithm (Johnson et al. 1974) are the simple and most widely used algorithms in the field of bin packing problems.

The first fit decreasing (FFD) algorithm is a simple approximation algorithm and it works as follows: sort the items in non-increasing order of sizes. Then starting packing with the first item in the list (which is the largest item) and place it into the bin with the lowest index, which it will fit this item while still meeting its capacity constraint. Eilon and Christofides (1971) indicate that the performance of FFD algorithm is quite good compared to the results of the previous studies. The best fit decreasing (BFD) algorithm is slightly better approximation algorithm. It works almost the same to the first fit decreasing (FFD) algorithm. However, there is a difference in determining which bin that the item will be placed in. Where in the BFD algorithm will choose the bin with the highest load (fullest bin), provided it fits this item and without exceeding the bin capacity. Although both of FFD and BFD algorithms could be utilized in $O\ (n\ \log n)$ time, this is not promised to get optimal solutions. However, the worst case bounds for both of these algorithms is $\frac{11}{9} * N + 4$, where N is the optimal number of bins (Johnson et al. ,1974).The main weakness of the FFD and BFD algorithms lies in the deterioration of their performance in dealing with the difficult problems (the problems that their optimal solution need a totally filling for the most or all bins). Coffman et al. (1978) show that the obtained solutions from the FFD and BFD algorithms are usually need more bins than the ones of an optimal solution for the difficult problems (cited in Gupta and Ho,1999).

Eilon and Christofides (1971) propose an improvement algorithm for solving the bin packing problem with different objective functions (cited in Kumar et al. ,2003).

Coffman et al. (1987) assert that the first fit decreasing (FFD) algorithm provides an optimal solution for one-dimensional bin packing problem under a divisibility condition. On the other hand, Kang and Park (2003) show that this result is incorrect through giving an opposing example.

Martello and Toth (1990) describe several simple heuristics and use a reduction procedure ($MTRP$) and an exact algorithm (MTP) to solve the bin packing problem (BPP) (cited in Loh et al, 2008).

Falkenauer (1996) propose a hybrid grouping genetic algorithm to solve the bin packing problem (cited in Fleszar and Charalambous, 2011).

Scholl et al. (1997) improve a hybrid method by gathering tabu search with a branch-and-bound method. Schwerin and Wäscher (1999) improve the MTP of Martello and Toth (1990) and also found a new lower limits for the bin packing problem (BPP) that is derived from the cutting stock problem. However, a comprehensive review of approximation schemes for the bin packing problem is given by Coffman et al. (1997), as the most important points dealt with is the analysis of the worst case of the first fit decreasing (FFD) and the best fit decreasing (BFD) algorithms (cited in Loh et al, 2008).

Vance (1998) propose an exact algorithm for solving the bin packing problem and this algorithm is relied on the linear programming methods which are proposed by Gilmore and Gomory (1961). However, the running time of this algorithm is a bit slow which is negatively impact on its practical use (cited in Kang and Park, 2003).

Gupta and Ho (1999) introduce a minimal bin slack heuristic (MBS) heuristic to solve the one-dimensional bin packing problem, which is developed later by Fleszar and Hindi (2002). They show that their proposed algorithm outperforms both of first fit decreasing (FFD) algorithm and best fit decreasing (BFD) algorithm regarding the optimality of solutions in particular for the problems that called "difficult" problems.

Vanderbeck (1999) describe an exact algorithm which is based on column generation for the cutting stock problem and show that this algorithm could be used for some kinds of bin packing problem (BPP) (cited in Fleszar and Charalambous, 2011).

Chu and La (2001) investigate four greedy approximation algorithms to solve the one-dimensional bin packing problem and study their absolute worst-case performances. They show that the worst case for these algorithms are $2,\ 2,\ 3$ and $2 + \ln 2$ in succession.

Fekete and Schepers (2001) provide a new lower bounds for the bin packing problem that based on dual-feasible functions (cited in Fleszar and Charalambous, 2011).

Furthermore, Fleszar and Hindi (2002) introduce a number of heuristics which rely on MBS and a variable neighbourhood search metaheuristic (cited in Loh et al, 2008).

Valério de Carvalho (2002) improve an exact algorithm by using the branch and bound algorithm and investigate linear programming (LP) models for the bin packing problem and the cutting stock problem.

Fleszar and Hindi (2002) propose a number of algorithms to solve the one dimensional bin packing problem. Some of these algorithms relied on the minimal bin slack (MBS) heuristic that is proposed by Gupta and Ho (1999), while there is a one based on the variable neighbourhood search scheme. However, their most efficient algorithm compared to other existing methods is based on operating the modified version (MBS') of the minimal bin slack heuristic then followed it by the variable neighbourhood search metaheuristic.

The call bin completion algorithm for optimal bin packing is proposed by Korf (2002) in which considering the methods of packing each bin to be completed) instead of investigating the possible bins that each item could be packed into. It is showed that this algorithm is quicker than the existing optimal algorithms.

Kumar et al. (2003) propose an algorithm for solving the one-dimensional bin packing problem with additional constraints. They used this heuristic for a vehicle allocation problem where this heuristic show its superiority over the first fit decreasing (FFD) algorithm in terms of better performance and easily alteration with other constraints.

Ross et al., (2003) investigate an approach based on genetic algorithm (GA) to solve the bin packing problem. Caprara and Pferschy (2004, 2005) consider the performance of the worst-case of heuristics (cited in Dokeroglu and Cosar, 2014).

Bhatia and Basu (2004) present a multi-chromosomal grouping genetic algorithm for BPP. Levine and Ducatelle (2004) introduce a hybrid method that applies the ant colony optimization metaheuristic ($HACO-BP$), which hasa technique for a local search based on the dominance criterion from Martello and Toth (1990). Singh and Gupta (2007) introduce a new heuristic that combines a hybrid steady-state grouping genetic algorithm with a developed minimal bin slack algorithm of Fleszar and Hindi (2002). Additionally, evolutionary algorithms are considered in Poli et al. (2007) and Rohlfshagen and Bullinaria (2007). Crainic et al. (2007a,b) introduce better lower bounds and study their worst case performance (cited in Fleszar and Charalambous, 2011).

Rohlfshagen and Bullinaria (2007) improve an algorithm that adopted the theory of exon shuffling. Poli et al. (2007) present an algorithm with discrete item sizes in which the histogram of item-size is joined with the corresponding bin-gap histogram. Stawowy (2008) propose a non-specialized and non-hybridized algorithm which uses an adjusted permutation with separators encoding strategy, unique concept of separators movements over mutation, and separators removal as a strategy to reduce the size of problem (cited in Dokeroglu and Cosar, 2014).

Roy et al. (2008) study the behavior patterns through practical instances from an empirical study of bin packing heuristics.

Loh et al. (2008) introduce a new heuristic based on using the weight annealing (WA) for solving the one-dimensional bin packing problem (BPP). Their computational experiments show that this technique is superior to most other previous approaches in terms of the simplicity of the algorithm, the high quality of the obtained solutions and the quickness of the running time.

Correa and Epstein (2008) consider a bin packing with controllable item sizes, where is given list of pairs related to each item. These pairs comprise of a permitted size for the item and a nonnegative penalty for each pair. The objective is to choose a pair for each item which minimizing the total number of bins that required to place the sizes and the sum of penalties. They also provide an asymptotic polynomial time approximation scheme ($APTAS$) which uses bins sizes are a little larger than 1.

Gómez-Meneses and Randall (2009) consider a new evolutionary approach that applies the hybrid extremal optimization (HEO). This concept is about eradicating the weakest element of a population and then replacing it with another random element. However, this method contains a local search which relies on the strategy that is proposed by Falkenauer (1996) in order to enhance the quality of the packing. Lewis (2009) introduce an intuitive hill-climbing (HC) procedure which uses a simple improvement strategy relies on the dominance criterion in order to make the bins more full. This procedure gives positive solutions and its performance is better than some other algorithms that considered in (Falkenauer, 1996; Gupta and Ho, 1999) while still less than the best state-of-the-art algorithms (cited in Quiroz-Castellanos et al., 2015).

Khanafer et al. (2010) propose an outline for acquiring new dual feasible functions that depend on data. Memetic algorithms is also used for solving the one dimensional bin packing problem. In particular, one of these strategies is based on using separate individual learning or local improvement procedures (Le et al., 2009; Ong et al., 2006). Segura et al. (2011) consider a multi-objectivized memetic algorithm to solve the two-dimensional bin packing problem which runs faster than the existing genetic algorithms (cited in Dokeroglu and Cosar, 2014).

Fleszar and Charalambous (2011) study the bin-oriented heuristics ($BOHs$) for the one dimensional bin packing problem (BPP). In bin-oriented heuristics, the solutions are constructed by packing one bin at a time. Fleszar and Charalambous (2011) propose a controlling average weight method for items which packed by using bin-oriented heuristics and give reduction methods for bin-oriented heuristics. As well as, they provide an improvement heuristic rely on this strategy. Their results show that both of controlling average weight method and reduction methods provided improved solutions with better computational times of some bin-oriented heuristics.

Also, they indicate that the performance of the new improvement heuristic is better than other previous heuristics with respect to the average quality of the solution and processing time.

Alvim et al. (2004) use a highly effective hybrid improvement heuristic (HI_BP) to solve the bin packing problem (BPP) and show that its performance is extremely well (cited in Loh et al., 2008).

Dokeroglu and Cosar (2014) propose an island parallel grouping genetic algorithms ($GGAs$) which are robust tools for solving the one dimensional bin packing problem. Their findings indicate that these proposed algorithms are probably one of the best algorithms to solve the one dimensional bin packing problem because they give a high quality of solution and a reasonable computation time in comparison with the state-of-the-art heuristics.

Quiroz-Castellanos et al. (2015) propose a Grouping Genetic Algorithm with Controlled Gene Transmission ($GGA - CGT$) to solve the bin packing problem. This suggested algorithm is supported the transmission of the best genes of the chromosomes while still keeping the balance between the selective pressure and population diversity.

2.1.4. Other versions of the bin packing problem

The basic bin packing problem is extended to several areas in order to demonstrate the real world applications. Some examples of the problem extensions are the two-dimensional packing problem [Martello and Vigo (1998)] and three-dimensional packing problem [Martello et al. (2002)], determining bounds of different bin packing problems [Fekete et al. (2001), Fleszar et al. (2002), Labbe et al. (2003), etc.], and considering more additional constraints [Robb and Trietsch (1999), Ralphs et al. (2003), etc.]. However, the classical bin packing problem could also extended to address special constraints such as packing grouping of items and the maximum number of items per bin. Anily and Federgruen (1991) considered the packing problem in the case of items are combined into different groups. They used this procedure in vehicle routing problem and partitioning problems. Rhee (1993) investigated the packing problem with additional restrictions about the maximum permitted number of items for each bin. He proved in his study that the difference between the expected numbers of bins when the maximum number of items is two and the expected number of bins when the maximum number of items is three is of the order $\sqrt{n}$, where n is an independent random variables uniformly distributed over $[0, 1]$. He also indicated that the difference will be smaller in the case of considering higher values of the maximum number of items (cited in Kumar et al.,2003).

Also, Xavier and Miyazawa (2005) consider the class constrained shelf bin packing problem ($CCSBP$) which is aimed to pack the items in a minimum number of bins, where the items should be separated by a shelf division of size d, where d is non-negative values. They propose hybrid algorithms relied on the first fit (decreasing) and best fit (decreasing) algorithms and gave an asymptotic polynomial time approximation scheme ($APTAS$) for $CCSBP$ problem when there is a bound C for the different classes, where C is constant.

Moreover, Filippi (2007) address a bin packing problem with a fixed number of object weights (BPC) which is considered as a high-multiplicity version of the classical bin packing problem because each object has its own weight so it is required to deal with each objects separately. His analysis leads to obtain a new bound on the gap between the optimal values of this problem and the linear relaxation of its Gilmore–Gomory formulation.

Furthermore, Epstein et al. (2011) consider a new kind of online bin packing with conflicts as well as address both of online and semi-online versions of this problem.

In addition to that, Masson et al. (2013) propose an efficient multi-start iterated local search for packing problems ($MS-ILS-PPs$) algorithm for multi-capacity bin packing problems ($MCBPP$). Their findings indicate that this approach (which is based on simple neighborhoods) provides good solutions with respect to the quality and the computational time, this also applies even for large problem instances.

2.2. Variable sized bin packing problem ($VSBPP$)

2.2.1. Problem definition

The variable sized bin packing problem ($VSBPP$) is a generalization of the classical one-dimensional bin packing problem (BPP). In the variable sized bin packing problem, we have a set of items in which each item has a specified size and different types of bins, where the number of bins is unlimited. The aim is to pack a set of items into a minimum number of bins while still meeting the capacity constraint of each bin. The $VSBPP$ is also a NP-hard problem because BPP (which is a special case of $VSBPP$) is a NP-hard problem (Garey and Johnson, 1979 cited in Correia et al. , 2008).

2.2.2. Usage $VSBPP$ in real-world applications

The variable sized bin packing problem ($VSBPP$) also has a wide range of practical applications for example in loading problems and in machine scheduling.

The $VSBPP$ arises in loading truck problems in the case where just the weight is taken into account and where a several trucks is available, specifically more than one truck of every size/weight limit. The objective is to minimize the overall cost of the chosen trucks.

In the case of machine scheduling, the $VSBPP$ originates when there are a given number of tasks and different types of processors, where each job has a processing time value that is required for its implementation. The aim is to minimize the cost related to the processors that is used to schedule all the tasks (Correia et al. , 2008).

2.2.3. Related work

A number of previous studies have been considered the methods of approximation solutions for variable sized bin packing problem ($VSBPP$) and its variants. Friesen and Langston (1986) describe three approximation algorithms for solving the variable sized bin packing problem where it allowable only a fixed set of bin sizes and the cost of the obtained solution is the total sizes of used bins. Also they show their guarantee asymptotic worst-case performance bounds which are 2, 3/2 and 4/3 in succession. Murgolo (1987) obtain an asymptotic fully polynomial time approximation scheme ($AFPTAS$) for this problem (Cited in Haouari and Serairi, 2009).

Han et al. (1994) consider an optimization problem for the two-dimensional variable sized vector bin packing problem ($2-VSVBP$), where is given different types of bins (not identical bins). They propose three approaches: a greedy heuristic, a method based on simulated annealing and an exact algorithm. In addition to use a method based on linear programming to improve lower bounds.

Monacci (2002) suggest a branch-and-bound method to solve the variable sized bin packing problem ($VSBPP$). He assume in his study that for each bin, its cost is equal to its capacity and the amount of bins per bin type is equal to the total amount of items (cited in Correia et al., 2008).

The column generation strategies are considered in (Belov and Scheithauer, 2002 ; Alves and Valério de Carvalho, 2007) and are applied to solve the variable sized bin packing problem ($VSBPP$) and the classical bin packing problem (BPP) as well. Moreover, Pisinger and Sigurd (2005) develop these column generation techniques for solving the two-dimensional variable sized bin packing problem ($2-DVSVBP$) (cited in Correia et al., 2008).

In addition to those existing methods, exact methods have been also investigated for the variable sized bin packing problem ($VSBPP$)

by Monaci (2002), Belov and Scheithauer (2002), Alves and Valério de Carvalho (2007), and Haouari and Serairi (2009). Nevertheless, these proposed exact algorithms is not capable for solving the large problem instances because the $VSBPP$ is NP-hard (Cited in Haouari and Serairi, 2009).

Kang and Park (2003) propose two greedy algorithms where they are a different form of first fit decreasing algorithm (FFD) and best fit decreasing algorithm (BFD) respectively. They analyze the asymptotic worst-case performance of these algorithms in three specific cases regarding the divisibility of items weights and/or bins capacities. Firstly, when the sizes of items and the sizes of bins are divisible and show that the algorithms give optimal solutions. In the second case, when only the sizes of bins are divisible and prove that the algorithms give a solution whose value is less than $\frac{11}{9}\,z + 4\,\frac{11}{9}$. Finally, when the sizes of bins are not divisible and prove that the algorithms give a solution whose value is less than $\frac{3}{2}\,z + 1$ (where z is the value of an optimal solution).

Correia et al. (2008) consider in their study the utilization of a discretized formulation for solving the variable sized bin packing problem ($VSBPP$). They show that their proposed model after having some appropriate improvements gives better linear programming bounds and also this model could be used jointly with a commercial package in order to find $VSBPP$ optimal solution.

Haouari and Serairi (2009) propose and evaluate the performance of six heuristics and also develop a genetic algorithm for the one dimensional variable sized bin packing problem ($VSBPP$). Their results show that these heuristics which based on set covering performed well for large problem instances in terms of providing highly efficient solutions and taking short CPU times.

Hemmelmayr et al. (2012) propose a variable neighbourhood search metaheuristic to solve the variable sized bin packing problem ($VSBPP$). This algorithm is based on using the lower bounds and dynamic programming. They indicate that this approach is more likely to have a better results than the current state-of-the-art methods, in particular when it is used with large-scale instances.

The generalization of first fit decreasing (FFD) algorithm in multidimensional case makes it necessary to define the methods of measuring and comparing items due to the fact that the largest item will be chosen and placed into a bin in the classical first fit decreasing (FFD) algorithm (cited in Gabay and Zaourar, 2013). Panigrahy et al. (2011) use the DotProduct measure which defines the term "largest" as the item that maximizes the dot product between the vector of remaining capacities and the vector of demands for the item.

2.2.4. Other versions of the variable sized bin packing problem

In light of previous studies, there are other suggested variants of the variable sized bin packing problem ($VSBPP$) could be defined as well.

In the original version, there are unlimited number of bins available for each type of the bins (cf. Friesen and Langston, 1986 ; Murgolo, 1987; Chu and La, 2001 ;Monacci, 2002 ;Kang and Park, 2003) (cited in Hemmelmayr et al , 2012).

Also, Dawande et al. (2001) address the variable sized bin packing problem with new constraints, named the color constraints. In this problem, each item has colour and size and the objective is to minimize the number of used bins such that each bin should not contain more than p distinct colors, where p is a pre-determined positive integer.

By the way, Seiden et al. (2003) study the variable sized online bin packing problem and propose algorithms which give better upper bounds compared to the existing ones as well as introduce the first lower bounds for this problem.

Another different form is examined by Correia et al., (2008) and Crainic et al., (2011), where in that case, an upper bounds on the number of bins per bin type are considered.

In addition to that, Correia et al. (2008) describe the variable cost and sized bin packing problem ($VCSBPP$) in which they consider the economic attributes (bin costs) in addition to physical attributes for the purpose of making more distinction between this case and the other case where it is not necessary to have a correlation between the fixed costs of the bins and their capacity. Epstein and Levin (2008) provide an asymptotic polynomial time approximation scheme ($APTAS$) for the generalized problem. Crainic et al. (2011) introduce a heuristics algorithms for $VCSBPP$, which relies on the upper and lower bounds. Their findings prove that these algorithms are very effective for large problem instances as well. It is also show how the correlation between the bin costs and the bin volumes affects the quality of the solution. So, this approach compared with state-of-the-art methods is provided better solutions regards to the computational effort and solution accuracy.

Furthermore, Baldi et al. (2010) study a more general version of this problem, where other characteristics are added for instance required items and optional items which should be placed into the bins. Besides this, they consider that the number of bins per bin type have a lower bound (cited in Hemmelmayr et al., 2012).

2.3. Vector bin packing problem (VBP)

2.3.1. Problem definition

The Vector bin packing (VBP) problem or d-Dimensional vector packing ($d - DVP$) problem is introduced by Garey et al. (1976) which is a generalization of the classical bin packing problem. In this problem, a given set of items where each item is a d-dimensional vector with entries $\in [0,1]$. The objective is to pack the items into a minimum number of bins where the sum of the sizes of all packed items must be less than or equal to 1 (cited in Alves et al. , 2014).

2.3.2. Usage VBP in real-world applications

There are many important applications of the vector bin packing (VBP) problem, one of them is Data Placement problem which takes a place in a study by Shachnai and Tamir (2003). It is also used in a shared hosting platform which is aimed to allocate the jobs to servers, where each job requires a number of resources like a number of cycles per second, memory and bandwidth. Therefore, in this application the jobs represent the items, the servers are the bins and the number of resources is the dimension d (Stillwell et al, 2010 cited in Kao, 2008). Another application of the vector packing problem is in modelling the virtual machine placements for the cases when all the machines have an identical capacities (Lee et al., 2011; Panigrahy et al., 2011; Stillwell et al., 2010). However, by the development of this area over the previous years, the new machines become with different capacities. A generalization of the vector bin packing problem (VBP) called the variable size vector bin packing ($VSVBP$) problem is introduced by Gabay and Zaourar (2013). The new in this problem is that each bin has a tuple of capacities and the aim is to pack the items in a minimum number of bins used. The $VSVBP$ problem efficiently modelling the virtual machine placements with heterogeneous cluster (cited in Gabay and Zaourar, 2013).

2.3.3. Related work

The first asymptotic polynomial-time approximation scheme ($APTAS$) is provided by Fernandez de la Vega and Lueker (1981) in which their method was based on rounding. Then Karmarkar and Karp (1982) improved this algorithm to a $(1+\log^2\ )$ -OPT bound (Cited in Rao et al., 2010).

Maruyama et al. (1977) study a generalization of one dimensional bin packing heuristics within a general framework for vector bin packing problem. Kou and Markowsky (1977) investigate the lower and upper bounds in their study and indicate that for some generalized classical bin packing algorithms, the behavior ratio of worst case is larger than the dimension (d) (cited in Gabay and Zaourar, 2013).

Yao (1980) proved that a worst case performance ratio of any time algorithm $O(n \log n\ )$ is bigger than dimension (d).

There are several algorithms used in solving the vector bin packing problem starting with the simple greedy heuristics for example: First Fit, Best Fit, Worst Fit and Next Fit algorithms which were studied in Kou and Markowsky,1977 ; Maruyama et al.,1977 (cited in Stillwell et al., 2010)

Woeginger (1997) prove that there is no asymptotic polynomial time approximation scheme ($APTAS$) for the vector bin packing problem of higher dimension $(d \geq 2)$ (unless $P = NP$). Chekuri and Khanna (1999) show an $O(\ln d)$-approximation algorithm for the vector bin packing which is a polynomial-time for the case where d is constant. Bansal et al. (2006) improve this by a randomized $(\ln d + 1 + \varepsilon)$-approximation algorithm that runs in polynomial-time for any fixed $\varepsilon > 0$ and constant dimension d. As well as this approximation algorithm has been improved to extend to higher dimensions ($d \geq 2$) by Rao et al. (2010), their proposed algorithm is dependent on combining both of (near-) optimal solution of the linear programming relaxation and a greedy heuristic. Karger et al. (2007) show the existence of the polynomial approximation scheme to the randomly perturbed instances through using smoothing analysis for multidimensional vector bin packing problems.

Karp et al. (1984) consider in their study the vector bin packing problem where the size of all items are drawn independently from the uniform distribution over $[0,1]$. They prove the lower bounds on the expected wasted space in the optimal solution is $\Omega\left(n^{\frac{d-1}{d}}\right)$ for $d > 3$. Also, they propose a new algorithms called $VPACK$ that tries to place two objects in each bin, Since this heuristics shows a better usage of the bins where the wasted space is considered as a very little amount.

Spieksma (1994) study the two dimensional vector packing ($2-DVP$) problem and propose a heuristic relies on the first fit decreasing (FFD) algorithm to solve this problem. As well as examine the lower bounds for optimal solutions of the two-dimensional vector packing ($2-DVP$) problem and using these bounds in a branch-and-bound algorithm.

Chekuri and Khanna (1999) show that the 2-dimensional vector packing problem is APX-hard and It is a $d^{1/2-\epsilon}$ hardness of approximation, for any fixed $\epsilon > 0$.

Caprara and Toth (2001) analyze many lower bounds for 2-dimensional vector packing problem and prove that all of these lower bounds are dominated by the acquired lower bound from the huge linear programming relaxation. They propose exact algorithms and heuristic in order to obtain an optimal solutions. A two-dimensional vector packing is also used by Chang et al. (2005) in modelling the packing steel products problem, where there are special containers should be packaged steel products and they propose a heuristic algorithm for it.

Alves et al. (2014) propose new functions called vector packing dual-feasible functions to solve the two-dimensional vector packing problem which extend the concept of dual-feasible functions to the multidimensional case. They show that theses proposed functions accomplish a considerable improvements on the convergence of branch and-bound algorithms and provide strong lower bounds.

Shachnai and Tamir (2003) propose a polynomial-time approximation scheme ($PTAS$) for a subclass of instances for the vector bin packing problem. Caprara et al. (2003) prove in their study that for getting a $PTAS$ for d-DVP, the weight vectors of all items must be totally ordered.

The genetic algorithms are also considered for solving the vector packing problems that arise from resource allocation problems (Rolia et al., 2003 ; Gmach et al. , 2009; Gmach, 2009 cited in Stillwell et al., 2010).

Stillwell et al. (2010) propose and assess several algorithms for the resource allocation problem in shared hosting platforms. They point out that the chose pack vector packing algorithm which is proposed by Leinberger et al. (1999) has the best performance regards the running time, as it does not exceed a few seconds. They also show that this approach is working better than greedy algorithms, linear programming relaxations and a genetic algorithms. Therefore, the chose pack vector packing algorithm is considered as more effective.

Panigrahy et al. (2011) study a various variants of the first fit decreasing (FFD) algorithm for solving the vector bin packing problem and propose a geometric algorithm which has a better results than first fit decreasing (FFD) heuristics for sensible values of n and d. In addition to that, the number of bins used could be reduced by 10% through using this new geometric heuristics.

Patt-Shamir et al. (2012) study a multiple-choice vector bin packing which is another different form of bin packing problem where bins have various sizes and they propose an approximation algorithm with a rate ($\ln 2d + 1 + \varepsilon$) for any $\varepsilon > 0$.

3. Problem definition

3.1. Variable size vector bin packing problem $(VSVBP)$

3.1.1. Notations and formulation

Consider the following notation:

$I = \{1, \dots \dots \dots \dots \dots \dots \dots, N\}$	set of items
$J = \{1, \dots \dots \dots \dots \dots \dots \dots, n\}$	set of bins
$D = \{1, \dots \dots \dots \dots \dots \dots \dots, d\}$	the number of dimensions
x_{ji}	item i is packed in bin j $(i \in I, j \in J)$
y_j	bin j is used
c_j^k	capacity of bin j in dimension k
s_i^k	size of item i in dimension k

The $VSVBPP$ can be straightforwardly formulated as follows:

$$\text{Min} \sum_{j \in J} y_j \qquad (1)$$

Subject to

$$\sum_{i \in I} s_i^k \, x_{ji} \leq c_j^k \qquad \forall \; j \in J \;, \forall \; k \in D \qquad (2)$$

$$\sum_{j \in J} x_{ji} = 1 \qquad \forall \; i \in I \qquad (3)$$

$$x_{ji} \in \{0,1\} \qquad \forall \; j \in J \,, \forall \; i \in I \qquad (4)$$

$$y_j \in \{0,1\} \qquad \forall \; j \in J \qquad (5)$$

The objective function (1) minimizes the number of the bins used for packing all the given items. Inequalities (2) demonstrate the capacity constraints which state that the amount of items packed in the bin j in dimension k should not exceed its capacity for each bin j and dimension k while constraints (3), (4) and (5) ensure that each item i is packed to a bin j.

The optimization problem that we are addressing is a two-dimensional variable size vector packing problem ($2DVSVP$). This problem is, in fact, a special case of the variable size vector bin packing problem which was introduced by Michael and Zaourar (2013) when the dimension $d = 2$.

3.2. Two-dimensional variable size vector packing problem ($\mathbf{2-DVSVP}$)

A given list of items $I = \{1, \dots \dots \dots \dots \dots \dots \dots, N\}$ and each item $i \in I$ has size 1 and size 2 (a_i, b_i). Also the size 1 and the size 2 of the bins is A and B respectively. The aim is to pack the items into a minimum number of bins such that the total sum of a_i (size 1) of all the items which packed into the same bin should not exceed A. Likewise, the total sum of b_i (size 2) of all the items which packed into the same bin should not exceed B.

However, in order to meet the constraint that the entries $(a_i, b_i) \in [0,1]$ for each item $i \in I$, it is required to scale the capacities of the bins (and items) so that the capacity of the bins will end up with the 1 for all dimensions. Hence, this could be obtained through dividing the capacity of each item by the capacity of the bin in that dimension.

4. Methodology

In this section, we explain the different strategies that have been used in our study for solving the two-dimensional variable size vector bin packing problem $(2-D\ VSVBP)$ which is a special case of the variable size vector bin packing problem $(VSVBP)$ when the dimension $d = 2$. These strategies are based on the first fit (FF) algorithm with some new variants.

4.1. Strategy 1

This strategy is applied the simple first fit (FF) algorithm which is used to solve the classical bin packing problem (BPP) into variable size vector bin packing $(VSVBP)$ problem which is a multidimensional packing problem. We generalize this well-known algorithm in order to investigate its performance in this multidimensional problem. In this research we refer to the first fit (FF) algorithm by algorithm 1. Since the asymptotic approximation ratio of First Fit bin packing is equal to 1.7, Dosa (2007) proved the absolute approximation ratio for the first fit bin packing is exactly equal to 1.7.

Algorithm 1 works as follow:

Step 1: Start packing with the first item in the list.

Step 2: check the fitting condition [If the item did fit in the first bin] then place the item into the first bin. Otherwise, open a new bin and put the item within the new bin.

Step 3: move to the next item and do the same procedure in the step 2 until packing all the items.

Note that the open bins they keep open in the hope that the remaining spaces will be filled later by other items.

4.2. Strategy 2

The concept of this strategy is the same as the first fit decreasing (FFD) algorithm which is one of the simple algorithms that is used to solve the bin packing problems (Eilon and Christofides 1971, Johnson et al. 1974). Also, different variants of the first fit decreasing (FFD) algorithm is studied by Panigrahy et al. (2011) to solve the vector bin packing problem.

In the FFD algorithm the set of items is sorted in non-increasing order regards their sizes. However, in our case we are dealing with multidimensional (2 dimensions) so it is important to define how the largest items will be measured. Our approach is to propose two algorithms, where the first one takes into account one of the sizes to measure the largest items with respect to it while the other size does not have any effect, it is just dependent on the selected size and the second algorithm is vice versa. Hence, we dealt with each size separately.

Algorithm 2 works as follows:

Step 1: sort the set of items in non-increasing order regards their size1, where size1 is the size of the items in the first dimension.

Step 2: apply the first fit (FF) algorithm to pack the items.

Algorithm 3 works as follows:

Step 1: sort the set of items in non-increasing order regards their size2, where size2 is the size of the items in the second dimension.

Step 2: apply the first fit (FF) algorithm to pack the items.

4.3. Strategy 3

This strategy is based on random permutation of items vector and it includes one algorithm called algorithm 4. This approach is the same of the Random Fit (RF) algorithm which is a simple variant of the well-know first fit (FF) algorithm (Albers and Mitzenmacher,1998).

Algorithm 4 works as follows:

Step 1: randomize the items vector.

Step 2: apply the first fit (FF) algorithm to the new obtained items vector.

To explain the randomization of items vector more precisely, for example: if we have in the original version of the problem 10 types of items, and there is 2 pieces from each item type except item type 1 and item type 7 there are 5 pieces from these items type. Thus, the total number of items is 26. Therefore, the original items vector is $\{1,1,1,1,1,2,2,3,3,4,4,5,5,6,6,7,7,7,7,7,8,8,9,9,10,10\}$, however , in this version the items vector would be randomize in any way such as $\{10,2,5,1,8,5,1,1,7,3,4,9,10,7,7,6,2,8,7,3,9,1,1,4,6,7\}$. So, the number of each type of items still as before just the order of these items change randomly.

Note that this randomization of items is changed every time when the algorithm runs which leads to obtain different results in each running while the input instances of the problem are the same.

4.4. Strategy 4

The strategy 4 rely on various probabilities rules and it is similar to the strategy 3 in terms of that both of them are selected the items randomly in each run of the algorithm. In other words, different results will be obtained for the same problem at every run of the algorithm.

In this strategy, we have four algorithms named algorithm 5, algorithm 6, algorithm 7 and algorithm 8 respectively. These algorithms have the same procedure except that the probabilities rules are different in each version.

Algorithm 5 is defined as follows:

Step 1: calculate the probability p_i for each item type, in which the probability rule in this algorithm is defined as follows:

$$p_i = \frac{demand\ (i)}{\sum_{i \in I} demand\ (i)}$$

where

$demand\ (i)$ is the number of units (items) of item type i

I is a set of item types

Step 2: find the cumulative distribution function (CDF).

Step 3:

1. Generate a random number r between $[0,1]$
2. If [the value of item (i-1) in CDF $< r \leq$ the value of item (i) in CDF] then
   2.1. select item(i)

   where $i = 1,2, \dots \dots \dots, n$ and n is the number of item types
3. Check the availability of item (i)
   3.1. If the item (i) is still available then
      3.1.1. select item (i).
      3.1.2. remove the item (i) from the original set of items.
   3.2. Otherwise, If the item (i) ran out then go to the stage 1 in step 3.
4. Iterate this procedure (step 3) until the original set of items is empty.

Step 4: construct a new set of items by the selected items via the previous rule so that their sequence will be the order of the items in this new set.

Step 5: pack the new list of items by using the first fit (FF) algorithm.

Algorithm 6 is defined as follows:

It has the same steps as in the algorithm 5 but it uses another probabilities rule. Its probabilities rule is

$$p_i = \frac{demand\ (i) * size1\ \ (\mathrm{i})}{\sum_{i \in I}(demand\ (i) * size1\ \ (\mathrm{i}))}$$

where

$demand\ (i)$ is the number of units (items) of item type i

$size1\ (i)$ is the size of the item i in the first dimension.

I is a set of item types

Algorithm 7 is defined as follows:

It runs with the same procedure of algorithm 5 except that the probabilities rule in this algorithm is defined as:

$$p_i = \frac{demand\ (i) * size2\ \ (\mathrm{i})}{\sum_{i \in I}(demand\ (i) * size2\ \ (\mathrm{i}))}$$

where

$demand\ (i)$ is the number of units (items) of item type i

$size2\ (i)$ is the size of the item i in the second dimension.

I is a set of item types

Algorithm 8 is defined as follows:

This algorithm is also follows the same instructions as the previous algorithms (algorithm 5, algorithm 6 and algorithm 7). However, it uses different probabilities rule which is defined as:

$$p_i = \frac{demand\ (i) * \text{average (i)}}{\sum_{i \in I}(demand\ (i) * \text{average (i)})}$$

where

$demand\ (i)$ is the number of units (items) of item type i

$average\ (i) = \frac{size1\ (i) + size2\ (i)}{2}$

$size1\ (i)$ is the size of the item i in the first dimension.

$size2\ (i)$ is the size of the item i in the second dimension.

I is a set of item types

4.5. Strategy 5

The strategy 5 is also based on the first fit decreasing (FFD) algorithm and it is associated with strategies 1 and 2 in terms of that all of them are deterministic algorithms. Within this strategy we have two algorithms, we denote them by algorithm 9 and algorithm 10. Since these algorithms are deterministic algorithms, their output are always the same for the same input instances.

Algorithm 9 is defined as follows:

Step 1: calculate the average size for each type of items,

where

$$average\ (i) = \frac{size1\ (i) + size2\ (i)}{2}$$

$size1\ (i)$ is the size of the item i in the first dimension.

$size2\ (i)$ is the size of the item i in the second dimension.

Step 2: sort the list of items in non-increasing order of their averages.

Step 3: pack the items using the first fit (FF) algorithm.

Algorithm 10 is defined as follows:

Step 1: calculate the weighted average size for each type of items,

where

$$weighted\ average\ (i) = \frac{a * size1\ (i) + b * size2\ (i)}{2}$$

$size1\ (i)$ is the size of the item i in the first dimension.

$size2\ (i)$ is the size of the item i in the second dimension.

a, b are the minimum number of bins that is required to pack the items of size 1 and size 2 respectively and they defined as follows:

$$a = \frac{\sum_{i \in I}(size1\ (i) * demand\ (i))}{Maxinum\ size1\ of\ the\ bin}$$

$$b = \frac{\sum_{i \in I}(size2\ (i) * demand\ (i))}{Maxinum\ size2\ of\ the\ bin}$$

$demand\ (i)$ is the number of units (items) of item type i

I is a set of item types

Step 2: sort the list of items in non-increasing order of their weighted averages.

Step 3: pack the items using the first fit (FF) algorithm.

The algorithm 9 is consider as a special case of algorithm 10 when both of a and b are equal to 1.

5. Experiments

We consider the variable size vector bin packing $(VSVBP)$ problem when the dimension $d = 2$. All the described algorithms in section 4 is experimented on two different sets of random instances in two cases: firstly, with an equal demand for each type of items. Secondly, with a random demand for each type of items. For each of these cases, the proposed algorithms will be run twice, in the first time with small-scale of instances and then with large-scale of instances. In this experiment we consider for each bin that the maximum size 1 and the maximum size 2 is 500 and 700 respectively. The instances that used in this experiment (set 1 and set 2 with different demands) is attached into the Appendix I. The proposed algorithms was implemented in Visual Basic (VB) and it is attached into the Appendix II.

5.1. Results:

In this section, we show the results of using the proposed strategies with two different data (set 1 and set 2) with different demand in each case. Regarding the deterministic strategies which are the strategy 1, the strategy 2 and the strategy 5 their results are obtained from the first run of the algorithm. On the other hand, the strategy 3 and the strategy 4 which are random strategies their results are obtained by run each algorithm ten times and take the average of the results.

The results are divided into two cases depending on the demand (the number of items for each type of items) either equal or random.

5.1.1. Results for case 1 (equal demand):

The given tables below (Table 1, Table 2, Table 3 and Table 4) show the obtained results from applying the suggested strategies into the set 1 and the set 2 of instances with an equal demand for both small-scale and large-scale of instances.

Strategies		Average Volume Size 1%	Average Volume Size 2%	Number of bins
1	Algorithm 1	79.45205	84.73581	73
2	Algorithm 2	79.45205	84.73581	73
	Algorithm 3	72.5	77.32143	80
3	Algorithm 4	83.97293	89.55734	71
4	Algorithm 5	80.72298	86.09126	72
	Algorithm 6	80.2337	85.56944	72
	Algorithm 7	81.3718	86.78322	71
	Algorithm 8	81.03751	86.42671	72
5	Algorithm 9	86.56716	92.32409	67
	Algorithm 10	82.35294	90.96639	68

Table 1: Set 1 of instances with small-scale and equal demand.

Strategies		Average Volume Size 1%	Average Volume Size 2%	Number of bins
1	Algorithm 1	81.76471	59.31373	102
2	Algorithm 2	87.78947	63.68421	95
	Algorithm 3	87.78947	63.68421	95
3	Algorithm 4	91.95835	66.70839	93
4	Algorithm 5	89.99079	65.28109	93
	Algorithm 6	86.63725	62.84837	96
	Algorithm 7	89.78326	65.13054	93
	Algorithm 8	88.48266	64.18706	94
5	Algorithm 9	92.66667	67.22222	90
	Algorithm 10	92.66667	67.22222	90

Table 2: Set 2 of instances with small-scale and equal demand.

Strategies		Average Volume Size 1%	Average Volume Size 2%	Number of bins
1	Algorithm 1	79.12688	84.38901	1466
2	Algorithm 2	80	85.3202	1450
	Algorithm 3	72.5	77.32143	1600
3	Algorithm 4	86.2528	91.98882	1346
4	Algorithm 5	82.77473	88.27945	1402
	Algorithm 6	81.60878	87.03597	1422
	Algorithm 7	83.25775	88.7946	1393
	Algorithm 8	83.23432	88.76961	1394
5	Algorithm 9	87.54717	93.36927	1325
	Algorithm 10	87.54717	93.36927	1325

Table 3: Set 1 of instances with large-scale and equal demand.

Strategies		Average Volume Size 1%	Average Volume Size 2%	Number of bins
1	Algorithm 1	82.20798	59.63529	2029
2	Algorithm 2	87.78947	63.68421	1900
	Algorithm 3	87.78947	63.68421	1900
3	Algorithm 4	91.91688	66.67831	1816
4	Algorithm 5	91.80082	66.59412	1817
	Algorithm 6	87.2086	63.26283	1913
	Algorithm 7	90.6516	65.76045	1840
	Algorithm 8	88.92962	64.5113	1876
5	Algorithm 9	92.66667	67.22222	1800
	Algorithm 10	92.66667	67.22222	1800

Table 4: Set 2 of instances with large-scale and equal demand.

Firstly, comparing the results between set 1 and set 2 for both cases small-scale and large-scale. Data from Table 1 can be compared with the data in Table 2 which shows that their results are consistent in the first three best results that means in other words in the strategy 5 and the strategy 3. However, there are some differences in the rest of the results, in particular with the strategy 4 as the performance of some of their algorithms is different in both of the set 1 and set 2. This also applies when comparing Table 3 with Table 4 as in this case (large-scale instances) the differences in the performance of the algorithms of strategy 4 is clearer.

Turning to compare the results of the small-scale instances with large-scale instances for each data set. In the set 2, it can be seen from the Table 2 and Table 4 that there is no differences between the results of the small-scale instances and the large-scale instances regards the order of superiority algorithms starting with the algorithms 9 and 10 which give the best results until the algorithm 1 which gives the worst results. In other words, the superior algorithms with the small-scale instances are still superior with the large-scale instances at the same order which is consider as a good indicator. As well as, in the set 1 as shown in Table 1 and Table 3 the order of superiority algorithms is the same in both small-scale and large-scale instances except that the algorithm 6 which gives the fourth-best result in the small-scale instances while in the large- scale its order in terms of superiority is the sixth.

Overall of case 1, the main observations that can be seen from Tables (1,2,3,4) above that the best results are obtained by the algorithm 9 and the algorithm 10 . Moreover, the algorithm 4 provides a roughly good result (the second best result). On the other hand, the algorithm 1 and the algorithm 3 give the worst result in set 2 and set 1 respectively.

5.1.2. Results for case 2 (random demand):

The next four tables (Table 5, Table 6, Table 7 and Table 8) show the results of using the proposed strategies into the data set 1 and the data set 2 with a random demand for items in both the small-scale and the large-scale of instances.

Strategies		Average Volume Size 1%	Average Volume Size 2%	Number of bins
1	Algorithm 1	79.45205	84.73581	73
2	Algorithm 2	79.45205	84.73581	73
	Algorithm 3	72.5	77.32143	80
3	Algorithm 4	83.97293	89.55734	71
4	Algorithm 5	80.72298	86.09126	72
	Algorithm 6	80.2337	85.56944	72
	Algorithm 7	81.3718	86.78322	71
	Algorithm 8	81.03751	86.42671	72
5	Algorithm 9	86.56716	92.32409	67
	Algorithm 10	82.35294	90.96639	68

Table 5: Set 1 of instances with small-scale and random demand.

Strategies		Average Volume Size 1%	Average Volume Size 2%	Number of bins
1	Algorithm 1	79.47115	57.3489	104
2	Algorithm 2	89.83696	64.82919	92
	Algorithm 3	88.87097	64.1321	93
3	Algorithm 4	91.90362	66.39077	92
4	Algorithm 5	89.57032	64.63678	92
	Algorithm 6	85.95516	62.02797	96
	Algorithm 7	90.2426	65.12192	92
	Algorithm 8	89.18904	64.36163	93
5	Algorithm 9	91.83333	66.26984	90
	Algorithm 10	91.83333	66.26984	90

Table 6: Set 2 of instances with small-scale and random demand.

Strategies		Average Volume Size 1%	Average Volume Size 2%	Number of bins
1	Algorithm 1	77.61111	67.81746	1800
2	Algorithm 2	82.17647	71.80672	1700
	Algorithm 3	77.61111	67.81746	1800
3	Algorithm 4	84.49777	73.8351	1654
4	Algorithm 5	81.4854	71.20286	1715
	Algorithm 6	82.07122	71.71475	1702
	Algorithm 7	82.0424	71.68957	1703
	Algorithm 8	82.8745	72.41667	1686
5	Algorithm 9	84.66667	73.98268	1650
	Algorithm 10	84.66667	73.98268	1650

Table 7: Set 1 of instances with large-scale and random demand.

Strategies		Average Volume Size 1%	Average Volume Size 2%	Number of bins
1	Algorithm 1	80.72727	54.38312	2200
2	Algorithm 2	82.60465	55.64784	2150
	Algorithm 3	84.57143	56.97279	2100
3	Algorithm 4	86.59192	58.33392	2052
4	Algorithm 5	85.9486	57.90054	2066
	Algorithm 6	83.82443	56.46956	2119
	Algorithm 7	85.50929	57.6046	2077
	Algorithm 8	84.00187	56.5891	2115
5	Algorithm 9	86.63415	58.36237	2050
	Algorithm 10	86.63415	58.36237	2050

Table 8: Set 2 of instances with large-scale and random demand.

Firstly, we compare the results of each data set in both cases (small-scale and large-scale). From Table 5 and Table 6 we can see that best results is given by the strategy 5 and then followed by the strategy 3 in terms of better results. Whereas the performance of strategies 1, 2 and 4 is different between data set 1 and data set 2. Similarly for the large-scale of instances, so that Table 7 and Table 8 have the same trend but we observe that there are more differences in the performance of strategy 2 between the two sets (set 1 and set 2) in which the algorithm 2 gives better results in data set 1 while the algorithm 3 provides better results in data set 2.

Secondly, we turn to compare the results of the small-scale instances and the large-scale instances for each set (set 1 and set 2) in the case of random demand.

In set 1, as can be seen from Table 5 and Table 7 that the strategy 5 outperform the other strategies in which their algorithms provide solutions that used a number of bins less than other strategies. Also, the strategy 3 still the second strategy that gives good solutions. In contrast, the performance of strategy 4 is varied between the small-scale and large-scale. For example, the algorithm 5 and the algorithm 7 gives better results with small-scale of instances while the algorithm 8 performs better with the large-scale of instances. As well as, the performance of algorithm 2 (which is within strategy 2) is better with the large-scale of instances.

Concerning set 2, Table 6 and Table 8 present the results of the data set 2 with random demand in small-scale and large-scale of instances respectively. The results of strategy 5 are still the dominant results throughout all the strategies. However, the performance of other algorithms is similar in both small and large instances except the algorithm 2 which gives the best fourth solution with small-scale of instances whereas it gives the ninth solution with the large-scale of instances.

Therefore, in this case the results suggest outperformed of the strategy 5, as well as a reasonable performance of the strategy 3. On the other hand, the performance of the strategy 1, the strategy 2 and the strategy 4 is various between the tables.

5.1.3. Summary of the results:

Summarising we can say that the strategy 5 gives the best results throughout all of the cases in both set 1 and set 2. However, the strategy 3 also gives a reasonable results in solving the variable size vector bin packing ($VSVBP$) problem.

5.2. Discussion

5.2.1. The superiority of strategy 5

The superiority of strategy 5 in our computational results is probably due to the algorithm 9 and algorithm 10 that are included in this strategy are taking into account both of size 1 and size 2 in the same time. In other words, the standards adopted by these algorithms is not biased to a certain size (dimension) without the other.

The observed difference between the algorithm 9 and the algorithm 10 in this study was not significant. However, it was expected to surpass the algorithm 10 even albeit slightly but we found the opposite. In Table 5, it has shown that the algorithm 10 packed the items in 68 bin while the algorithm 9 packed the same items in 67 bin. As we indicated that this difference is not great but it was expected that this superiority is in favour of the algorithm 10 because it is based on the weighted average.

Strategy 5 has another important advantage that their algorithms are deterministic algorithms so they give the same output even when the algorithms run several times. To illustrate the importance of this property for example, in the case where the strategy 3 gives the same obtained results from the strategy 5, then the preference will be for strategy 5 because their output is constant while the output of the strategy 3 changeable in each time we run the algorithm since it is based on randomization. Except in the case that the worst solution for strategy 3 is still better than the solution of strategy 5 therefore the strategy 3 is better in this case.

5.2.2. The relatively good performance of Strategy 3

The algorithm 4 (which is within the strategy 3) gives satisfactory results to some extent, due to it based on randomizing the items vector. Therefore, it arranges the items randomly and pack them in bins, this method is not like any of the deterministic methods that packing all the items of the selected type before moving to another type of items.

The randomization property is quite good because sometimes one or more of the items type have large size in one (or more) of its dimension. So it cannot be placed with another item of the same type and this makes the algorithm opens many of the new bins. In particular, when this type of items occurs as a one of the last items in the list. In this case, there is a less chance in having items with small sizes that could be placed with those large items in the same bin. On the other hand, the property of randomizing the items vector is more likely to reduce the number of bins because it puts the items in random order which will probably result in increasing the utilization of the bins used as we can see in Table 6 that the average utilization of size 1 and the average utilization of size 2 for the strategy 3 is 91.90362 and 66.39077 respectively which is better than the average utilization of strategy 5.

The principle which this algorithm is dependent on it (randomizing the vector items) gives different results in each run of the algorithm and this probably consider as a negative point for this approach. However, we could run the algorithm for several times and take the average of the obtained results, as well as taking into account the best solution and the worst solution of the obtained results.

5.2.3. The worst results

It is expected that the worst result will be by the algorithm 1 (first fit (FF) algorithm) because it is packing the items based on a very simple rule and it does not take into account any of the dimensions of the problem. It is packing all the items of the first type in the given set, then moving to the followed type of items and so on until packing all the set. However, we noted that the algorithm 3 (which is within strategy 2) gives the worst results in the data set 1, which is worse than the results of the algorithm 1 (first fit (FF) algorithm).

The reason behind the algorithm 3 gives the worst results in the data set 1 is that the items of type 5 has the smallest size regards size 2 and the largest size in terms of size 1 which leads the algorithm 3 to put the items of this type as the last items for packing. To explain in more detail, the items of type 5 have a large size (475) in size 1 and that the maximum size 1 is 500 per bin this leads to open new bin for each of these items because the bin cannot hold two items of this type as well as this type is the last type packed in this algorithm so there is no other types of items will fit with them in the same bin such as items of type 3 or 10 because these items placed before the items of type 5. This is the cause why the algorithm 3 uses more bins than in the algorithm 1.

5.2.4. The different performance of strategy 4

In general, by comparing the results we find that the performance of the strategy 4 is variable and its results usually in the middle, so are not good as the obtained results by the strategy 5 and are not bad as the results of strategy 1. In addition, as we indicated previously that both of the strategy 3 and the strategy 4 based on the randomization, but the results indicate that the performance of strategy 3 is superior to the performance of strategy 4 in all cases, as well as the performance of strategy 3 is constant, in other words, it consider as the second-best strategy for all cases. However, we did not expect this performance of the strategy 4, especially for the algorithm 8 which its probability rule rely on the average and the demand, so it was expected that the algorithm 8 gives good results because it takes into account all the dimensions of the problem and the demand as well.

6. Conclusion

6.1. Summary

In this study, we consider a special case of the variable size vector bin packing $(VSVBP)$ problem when the dimension $d = 2$. The $VSVBP$ problem is a generalization of the vector bin packing (VBP) problem. At the present, the $VSVBP$ is very useful in modeling many real-world applications because in recent years several real-life problems have a number of incomparable variables that are required to be consider at the same time whereas the variable size vector bin packing $(VSVBP)$ problem takes into account the multidimensionality so this makes this type of problem capable to deal with those applications. We propose five different strategies that are based on the well-known first fit (FF) algorithm and with some new variants for the variable size vector bin packing problem in order to minimize the number of bins used for packing a given set of items. These proposed algorithms are easy to implement and their running time is fast. The obtained results show that the algorithms 9 and 10 in the strategy 5 which are based on the average size of items and the weighted average size of items respectively produce the best solutions compared with the other proposed strategies, even for large-scale instances of both data sets. However, the strategy 3 which is rely on randomizing the items vector also gives a reasonable solutions in all the discussed cases.

6.2. Limitation

The most important limitation lies in the fact that this study did not take into account the minimum space needed between each pair of adjacent items which is probably required in some practical applications. So, this assumption was not addressed in this study.

6.3. Recommendations

In the future, it is strongly recommended to do further investigation and experimentation on the impact of the number of items for each item type into the suggested algorithms in this study. For example, in the case where there are more large items or more small items (after selecting a certain criteria for measuring the large and small) and it would be interesting to compare the findings. Further research could also be conducted to determine the effectiveness of the proposed strategies in solving the variable size vector bin packing problem when the dimension $d > 2$.

Glossary

$\boldsymbol{C\&P}$	Cutting and packing problems
$\boldsymbol{BPP}$	Bin packing problem
$\boldsymbol{VBP}$	Vector bin packing problem
$\boldsymbol{d-DVP}$	d-Dimensional vector packing problem
$\boldsymbol{VSVBP}$	Variable size vector bin packing problem
$\boldsymbol{2-DVSVBP}$	Two - dimensional variable size vector bin packing problem
$\boldsymbol{VCSBPP}$	Variable cost and sized bin packing problem
$\boldsymbol{CCSBP}$	Class constrained shelf bin packing problem
$\boldsymbol{BPC}$	Bin packing problem with a fixed number of object weights
$\boldsymbol{PTAS}$	Polynomial-time approximation scheme
$\boldsymbol{APTAS}$	Asymptotic polynomial time approximation scheme
$\boldsymbol{FF}$	First fit algorithm
$\boldsymbol{FFD}$	First fit decreasing algorithm
$\boldsymbol{BFD}$	Best fit decreasing algorithm
$\boldsymbol{MBS}$	Minimal bin slack heuristic
$\boldsymbol{LP}$	Linear programming
$\boldsymbol{GA}$	Genetic algorithm
$\boldsymbol{HACO-BP}$	Ant colony optimization metaheuristic

WA	Weight annealing
HEO	Hybrid extremal optimization
HC	Hill-climbing
$BOHs$	Bin-oriented heuristics
HI_BP	Hybrid improvement heuristic
$GGAs$	Grouping genetic algorithms
$GGA-CGT$	Grouping Genetic Algorithm with Controlled Gene Transmission
$MS-ILS-PPs$	Multi-start iterated local search for packing problems
$MCBPP$	Multi-capacity bin packing problem

References

Albers, S. and Mitzenmacher, M. Average-case analyses of first fit and random fit bin packing. In *Proc. Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, pages 290–299, Philadelphia*, 1998. Society for Industrial and Applied Mathematics.

Alves, C. and Valério de Carvalho, J. (2007). Accelerating column generation for variable sized bin-packing problems. *European Journal of Operational Research*, 183(3), pp.1333-1352.

Alves, C., de Carvalho, J., Clautiaux, F. and Rietz, J. (2014). Multidimensional dual-feasible functions and fast lower bounds for the vector packing problem. *European Journal of Operational Research*, 233(1), pp.43-63.

Bansal, N., Caprara, A. and Sviridenko, M. “Improved Approximation Algorithms for Multidimensional Bin Packing Problems,” Proc. IEEE Symp. Foundations of Computer Science, pp. 697-708, 2006**.**

Caprara, A. and Toth, P. (2001). Lower bounds and algorithms for the 2-dimensional vector packing problem. *Discrete Applied Mathematics*, 111(3), pp.231-262.

Caprara, A., Kellerer, H. and Pferschy, U. (2003). Approximation schemes for ordered vector packing problems. *Naval Research Logistics*, 50(1), pp.58-69.

Chang, S., Hwang, H. and Park, S. (2005). A two-dimensional vector packing model for the efficient use of coil cassettes. *Computers & Operations Research*, 32(8), pp.2051-2058.

Chekuri C, Khanna S (1999) On multi-dimensional packing problems. In: Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, SODA '99, pp 185-194.

Chu, C. and La, R. (2001). Variable-Sized Bin Packing: Tight Absolute Worst-

Case Performance Ratios for Four Approximation Algorithms. *SIAM J. Comput.*, 30(6), pp.2069-2083.

Coffman, E., Garey, M. and Johnson, D. (1987). Bin packing with divisible item sizes. *Journal of Complexity*, 3(4), pp.406-428.

Correa, J. and Epstein, L. (2008). Bin packing with controllable item sizes. *Information and Computation*, 206(8), pp.1003-1016.

Correia, I., Gouveia, L. and Saldanha-da-Gama, F. (2008). Solving the variable size bin packing problem with discretized formulations. *Computers & Operations Research*, 35(6), pp.2103-2113.

Crainic, T., Perboli, G., Rei, W. and Tadei, R. (2011). Efficient lower bounds and heuristics for the variable cost and size bin packing problem. *Computers & Operations Research*, 38(11), pp.1474-1482.

Dawande, M., Kalagnanam, J. and Sethuraman, J. (2001). Variable Sized Bin Packing With Color Constraints. *Electronic Notes in Discrete Mathematics*, 7, pp.154-157.

Dokeroglu, T. and Cosar, A. (2014). Optimization of one-dimensional Bin Packing Problem with island parallel grouping genetic algorithms. *Computers & Industrial Engineering*, 75, pp.176-186.

Eilon, S. and Christofides, N. (1971). The Loading Problem. *Management Science*, 17(5), pp.259-268.

Epstein, L. and Levin, A. (2008). An APTAS for Generalized Cost Variable-Sized Bin Packing. *SIAM J. Comput.*, 38(1), pp.411-428.

Epstein, L., Favrholdt, L. and Levin, A. (2011). Online variable-sized bin packing with conflicts. *Discrete Optimization*, 8(2), pp.333-343.

Filippi, C. (2007). On the bin packing problem with a fixed number of object weights. *European Journal of Operational Research*, 181(1), pp.117-126.

Fleszar, K. and Charalambous, C. (2011). Average-weight-controlled bin-oriented heuristics for the one-dimensional bin-packing problem.

European Journal of Operational Research, 210(2), pp.176-184.

Fleszar, K. and Hindi, K. (2002). New heuristics for one-dimensional bin-packing. *Computers & Operations Research*, 29(7), pp.821-839.

Gabay, M. and Zaourar, S. Variable size vector bin packing heuristics - Application to the machine reassignment problem. *HAL preprint hal-00868016*, Sept. 2013.

Gupta, J. and Ho, J. (1999). A new heuristic algorithm for the one-dimensional bin-packing problem. *Production Planning & Control*, 10(6), pp.598-603.

Gyrorgy Dosa. The tight bound of first fit decreasing bin-packing algorithm is F F D(I) ≤ 11/9OP T(I)+6/9. In *Combinatorics, Algorithms, Probabilistic and Experimental Methodologies*, pages 1–11. Springer, 2007.

Han, B., Diehr, G. and Cook, J. (1994). Multiple-type, two-dimensional bin packing problems: Applications and algorithms. *Annals of Operations Research*, 50(1), pp.239-261.

Haouari, M. and Serairi, M. (2009). Heuristics for the variable sized bin-packing problem. *Computers & Operations Research*, 36(10), pp.2877-2884.

Hemmelmayr, V., Schmid, V. and Blum, C. (2012). Variable neighbourhood search for the variable sized bin packing problem. *Computers & Operations Research*, 39(5), pp.1097-1108.

Johnson, D., Demers, A., Ullman, J., Garey, M. and Graham, R. (1974). Worst-Case Performance Bounds for Simple One-Dimensional Packing Algorithms. SIAM J. Comput., 3(4), pp.299-325.

Kao, M. (2008). *Encyclopedia of algorithms*. New York, NY: Springer.

Kang, J. and Park, S. (2003). Algorithms for the variable sized bin packing problem. *European Journal of Operational Research*, 147(2), pp.365-372.

Karger, D. and Onak, K. Polynomial approximation schemes for smoothed and random instances of multidimensional packing problems. In *SODA'07: the 18th annual ACM-SIAM symposium on Discrete algorithms*, pages 1207–1216, 2007.

Karp, R. ,Luby, M. and Marchetti-Spaccamela, A. A probabilistic analysis of multidimensional bin pa king problems. In *Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing*, pages 289-298, 1984.

Korf, R. 2002. A new algorithm for optimal bin packing. In *Eighteenth National Conference on Artificial Intelligence* (AAAI-02), 731–736.

Kumar, S., Rao, V. and Tirupati, D. (2003) "A heuristic procedure for one dimensional bin packing problem with additional constraints", IIMA Working Papers from Indian Institute of Management Ahmedabad, Research and Publication Department, No WP2003-11-02.

Loh, K., Golden, B. and Wasil, E. (2008). Solving the one-dimensional bin packing problem with a weight annealing heuristic. *Computers & Operations Research*, 35(7), pp.2283-2291.

Masson, R., Vidal, T., Michallet, J., Penna, P., Petrucci, V., Subramanian, A. and Dubedout, H. (2013). An iterated local search heuristic for multi-capacity bin packing and machine reassignment problems. *Expert Systems with Applications*, 40(13), pp.5266-5275.

Panigrahy, R. , Talwar, K. , Uyeda, L. and Wieder, U. (2011) Heuristics for vector bin packing. Tech. rep., Microsoft Research.

Patt-Shamir, B. and Rawitz, D. (2012). Vector bin packing with multiple-choice. *Discrete Applied Mathematics*, 160(10-11), pp.1591-1600.

Quiroz-Castellanos, M., Cruz-Reyes, L., Torres-Jimenez, J., G³ᶦmez S., C., Huacuja, H. and Alvim, A. (2015). A grouping genetic algorithm with controlled gene transmission for the bin packing problem. *Computers & Operations Research*, 55, pp.52-64.

Rao, C., Geevarghese, J. and Rajan, K., "Improved Approximation Bounds for Vector Bin Packing". Arxiv preprint arXiv:1007.1345, 2010.

Roy, N., Kinnebrew, J., Shankaran, N., Biswas, G. and Schmidt, D. Toward effective multi-capacity resource allocation in distributed real-time and embedded systems, in: Proceedings of the 11th Symposium on Object Oriented Real-Time and Distributed Computing, 2008.

Seiden, S., van Stee, R. and Epstein, L. (2003). New Bounds for Variable-Sized Online Bin Packing. *SIAM J. Comput.*, 32(2), pp.455-469.

Shachnai, H. and Tamir, T. Approximation schemes for generalized 2-dimensional vector packing with application to data placement. In *Proceedings of the 6th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, pages 167–177, 2003.

Spieksma, F. (1994). A branch-and-bound algorithm for the two-dimensional vector packing problem. *Computers and Operations Research*, 21(1), pp.19-25.

Stillwell, M., Schanzenbach, D., Vivien, F. and Casanova, H. (2010). Resource allocation algorithms for virtualized service hosting platforms. *Journal of Parallel and Distributed Computing*, 70(9), pp.962-974.

Valério de Carvalho, J. (2002). LP models for bin packing and cutting stock problems. *European Journal of Operational Research*, 141(2), pp.253-273.

Wأ¤scher, G., Hauنأner, H. and Schumann, H. (2007). An improved typology of cutting and packing problems. *European Journal of Operational Research*, 183(3), pp.1109-1130.

Woeginger, G. (1997). There is no asymptotic PTAS for two-dimensional vector packing. *Information Processing Letters*, 64(6), pp.293-297.

Xavier, E. and Miyazawa, F. (2005). A one-dimensional bin packing problem with shelf divisions. *Electronic Notes in Discrete Mathematics*, 19, pp.329-335.

Yao, A. (1980). New Algorithms for Bin Packing. *Journal of the ACM*, 27(2), pp.207-227.

Appendices

Appendix I

The small-scale of set 1 instances with equal demand

Item Type	Size 1	Size 2	Demand
1	50	200	10
2	175	150	10
3	25	600	10
4	450	550	10
5	475	20	10
6	250	500	10
7	425	450	10
8	40	80	10
9	70	60	10
10	20	245	10
11	120	575	10
12	350	450	10
13	50	175	10
14	175	25	10
15	225	250	10

The large-scale of set 1 instances with equal demand

Item Type	Size 1	Size 2	Demand
1	50	200	200
2	175	150	200
3	25	600	200
4	450	550	200
5	475	20	200
6	250	500	200
7	425	450	200
8	40	80	200
9	70	60	200
10	20	245	200
11	120	575	200
12	350	450	200
13	50	175	200
14	175	25	200
15	225	250	200

The small-scale of set 1 instances with random demand

Item Type	Size 1	Size 2	Demand
1	50	200	15
2	175	150	5
3	25	600	10
4	450	550	15
5	475	20	10
6	250	500	10
7	425	450	5
8	40	80	20
9	70	60	5
10	20	245	10
11	120	575	10
12	350	450	10
13	50	175	10
14	175	25	5
15	225	250	10

The large-scale of set 1 instances with random demand

Item Type	Size 1	Size 2	Demand
1	50	200	300
2	175	150	100
3	25	600	100
4	450	550	200
5	475	20	400
6	250	500	200
7	425	450	200
8	40	80	100
9	70	60	200
10	20	245	200
11	120	575	200
12	350	450	400
13	50	175	100
14	175	25	200
15	225	250	100

The small-scale of set 2 instances with equal demand

Item Type	**Size 1**	**Size 2**	**Demand**
1	350	275	10
2	475	300	10
3	225	175	10
4	400	100	10
5	200	160	10
6	480	620	10
7	375	275	10
8	275	300	10
9	450	550	10
10	20	50	10
11	225	150	10
12	300	440	10
13	225	370	10
14	150	400	10
15	20	70	10

The large-scale of set 2 instances with equal demand

Item Type	Size 1	Size 2	Demand
1	350	275	200
2	475	300	200
3	225	175	200
4	400	100	200
5	200	160	200
6	480	620	200
7	375	275	200
8	275	300	200
9	450	550	200
10	20	50	200
11	225	150	200
12	300	440	200
13	225	370	200
14	150	400	200
15	20	70	200

The small-scale of set 2 instances with random demand

Item Type	Size 1	Size 2	Demand
1	350	275	10
2	475	300	15
3	225	175	5
4	400	100	10
5	200	160	20
6	480	620	5
7	375	275	5
8	275	300	10
9	450	550	10
10	20	50	10
11	225	150	10
12	300	440	10
13	225	370	10
14	150	400	15
15	20	70	5

The large-scale of set 2 instances with random demand

Item Type	Size 1	Size 2	Demand
1	350	275	300
2	475	300	300
3	225	175	200
4	400	100	200
5	200	160	100
6	480	620	200
7	375	275	300
8	275	300	200
9	450	550	100
10	20	50	300
11	225	150	200
12	300	440	250
13	225	370	200
14	150	400	100
15	20	70	50

Appendix II

This is the class that would be called from all the algorithms (clsBin)

```
Option Explicit

Private size As Double
Private size2 As Double

Private LastUsedPlace As Double

Public Property Get sizeOfBin() As Double
    sizeOfBin = size
End Property

Public Property Get size2OfBin() As Double
    size2OfBin = size2
End Property

Public Property Get LastUsedPlaceOfBin() As Double
   LastUsedPlaceOfBin = LastUsedPlace
End Property
Public Property Let updateSize(value As Double)
    size = value
End Property

Public Property Let updateSize2(value As Double)
    size2 = value
End Property

Public Property Let updateLastUsedPlace(value As Double)
    LastUsedPlace = value
End Property
```

This is the configuration code of the algorithm 1

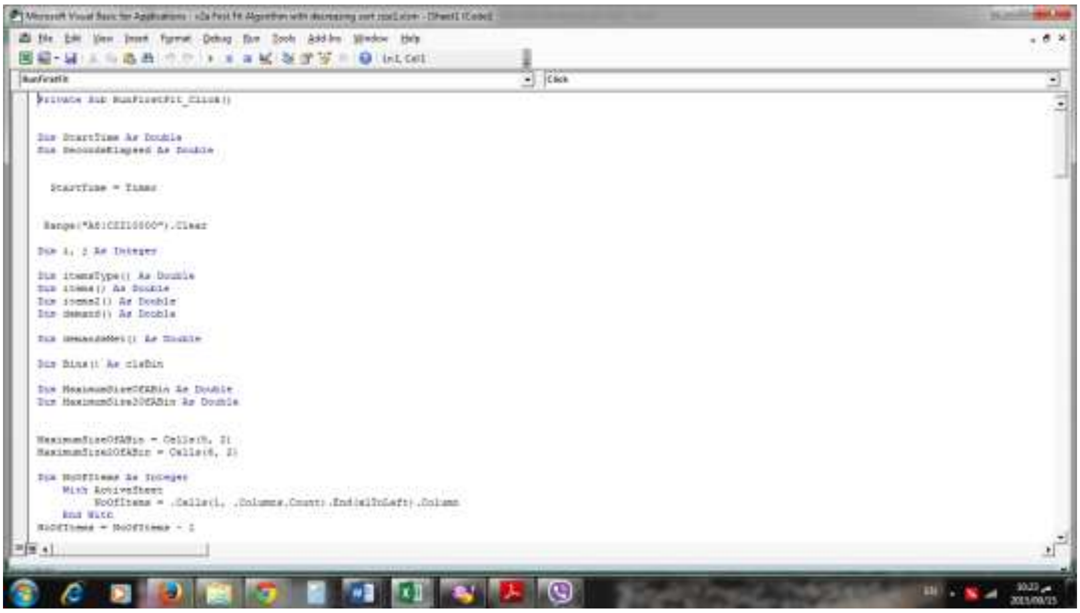
```
Private Sub RunFirstFit_Click()

Dim StartTime As Double
Dim SecondsElapsed As Double

StartTime = Timer

Range("A8:ZZ10000").Clear

Dim i, j As Integer
Dim items() As Double
Dim items2() As Double
Dim demand() As Double

Dim demandsMet() As Double

Dim Bins() As clsBin

Dim MaximumSizeOfABin As Double
Dim MaximumSize2OfABin As Double

MaximumSizeOfABin = Cells(5, 2)
MaximumSize2OfABin = Cells(6, 2)

Dim NoOfItems As Integer
    With ActiveSheet
         NoOfItems = .Cells(1, .Columns.Count).End(xlToLeft).Column
    End With
NoOfItems = NoOfItems - 1

ReDim items(NoOfItems)
ReDim items2(NoOfItems)
ReDim demand(NoOfItems)


Dim foundAplaceForTheItem As Boolean


Cells(8, 1) = "Bin 1"
Cells(7, 2) = "Volume Size 1 Perc%"
Cells(7, 3) = "Volume Size 2 Perc%"

For i = 0 To UBound(items) - 1

  items(i) = Cells(2, i + 2)


  items2(i) = Cells(3, i + 2)
  demand(i) = Cells(4, i + 2)

Next i

Dim SumDemands As Double

SumDemands = 0

For i = 0 To UBound(items) - 1

  SumDemands = SumDemands + demand(i)

Next i
```

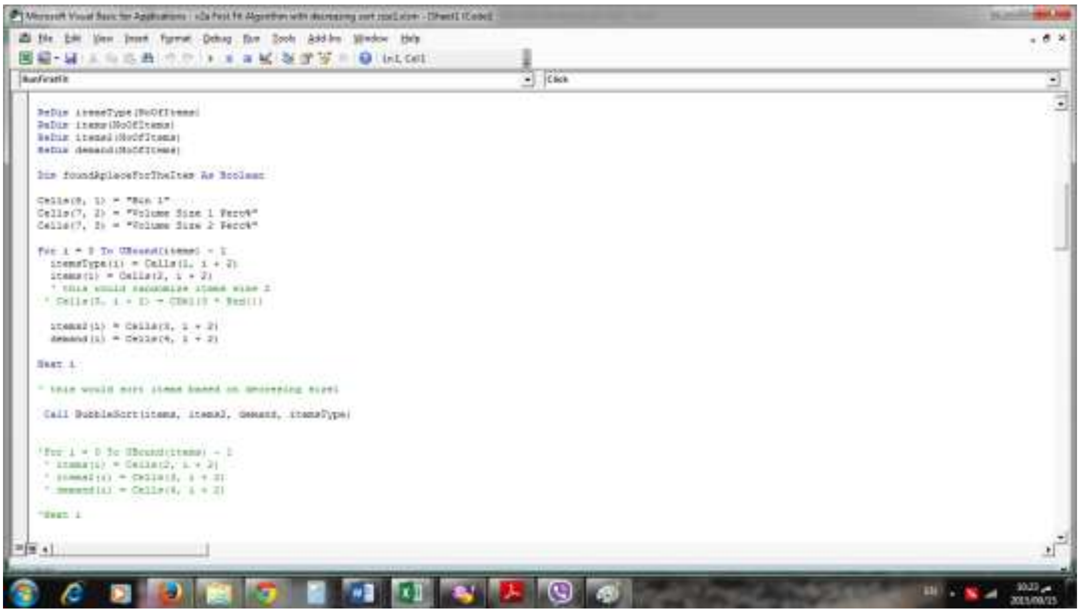
```
ReDim Bins(SumDemands) As clsBin

For i = 0 To SumDemands
   Set Bins(i) = New clsBin
   Bins(i).updateLastUsedPlace = 2
   Bins(i).updateSize = 0
   Bins(i).updateSize2 = 0

Next i

Dim LastUsedBin As Double


LastUsedBin = 0
For i = 0 To UBound(items) - 1

  foundAplaceForTheItem = False


  For j = 0 To LastUsedBin
    If (Bins(j).sizeOfBin + items(i) <= MaximumSizeOfABin And Bins(j).size2OfBin + items2(i) <= MaximumSize2OfABin) Then

       Bins(j).updateSize = Bins(j).sizeOfBin + items(i)
       Bins(j).updateSize2 = Bins(j).size2OfBin + items2(i)
       Bins(j).updateLastUsedPlace = Bins(j).LastUsedPlaceOfBin + 1
       Cells(j + 8, Bins(j).LastUsedPlaceOfBin + 1) = i + 1
       foundAplaceForTheItem = True
       Exit For
    End If

  Next j


   If (Not foundAplaceForTheItem) Then

     LastUsedBin = LastUsedBin + 1
     Cells(LastUsedBin + 8, 1) = "Bin " & CStr(LastUsedBin + 1)
     Bins(LastUsedBin).updateSize = Bins(LastUsedBin).sizeOfBin + items(i)
     Bins(LastUsedBin).updateSize2 = Bins(LastUsedBin).size2OfBin + items2(i)
     Bins(LastUsedBin).updateLastUsedPlace = Bins(LastUsedBin).LastUsedPlaceOfBin + 1

     Cells(LastUsedBin + 8, Bins(LastUsedBin).LastUsedPlaceOfBin + 1) = i + 1
     foundAplaceForTheItem = True


   End If
   ' the following is necessary in case demand will be count and handled whenever met (no randomizing vector)
   demand(i) = demand(i) - 1
   If (demand(i) > 0) Then
    i = i - 1
   End If

Next i
```

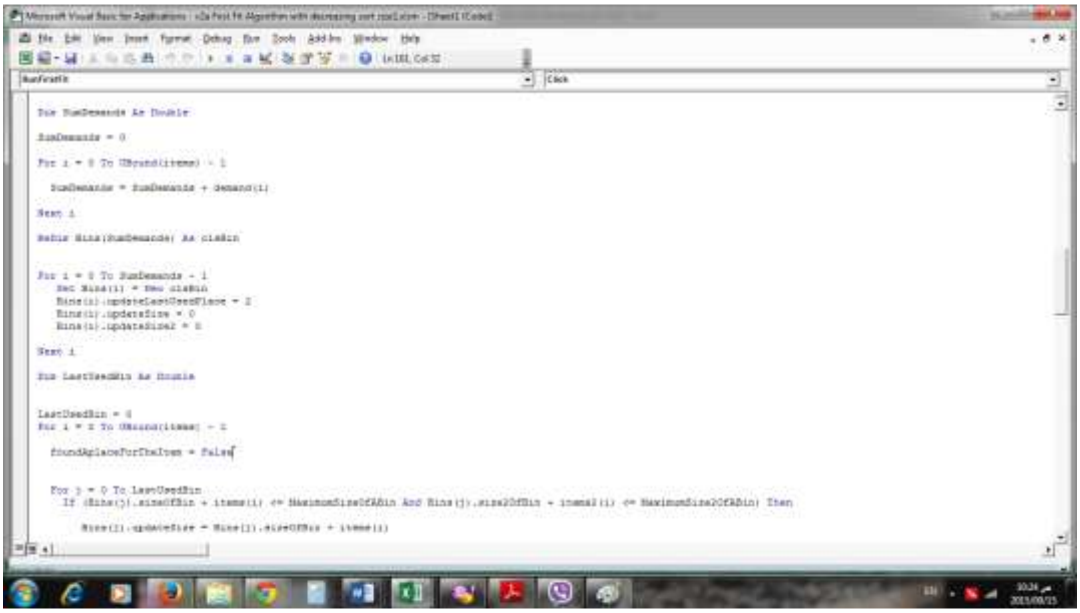
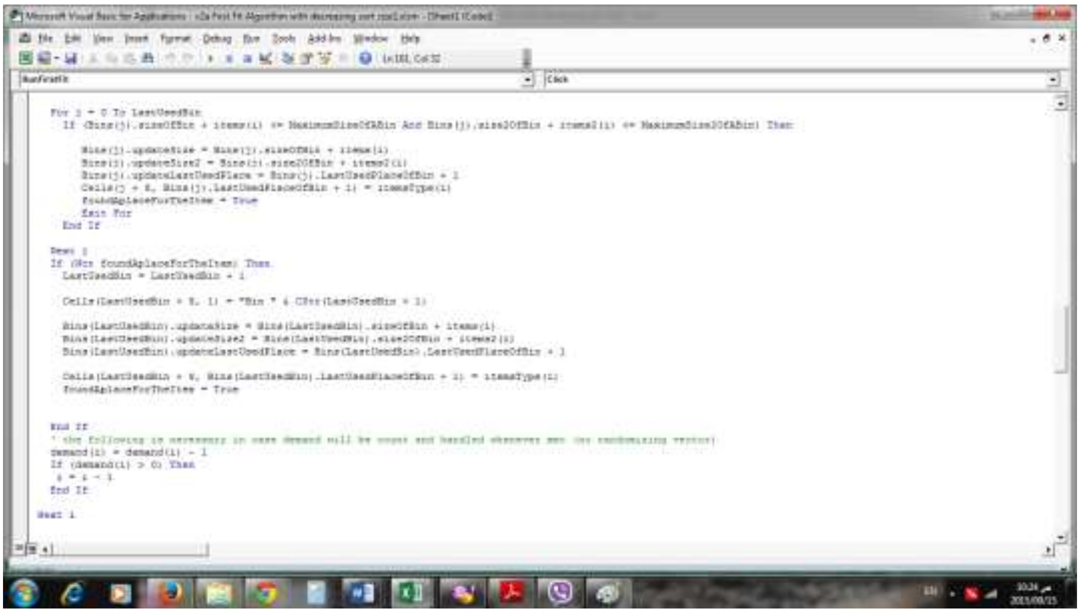
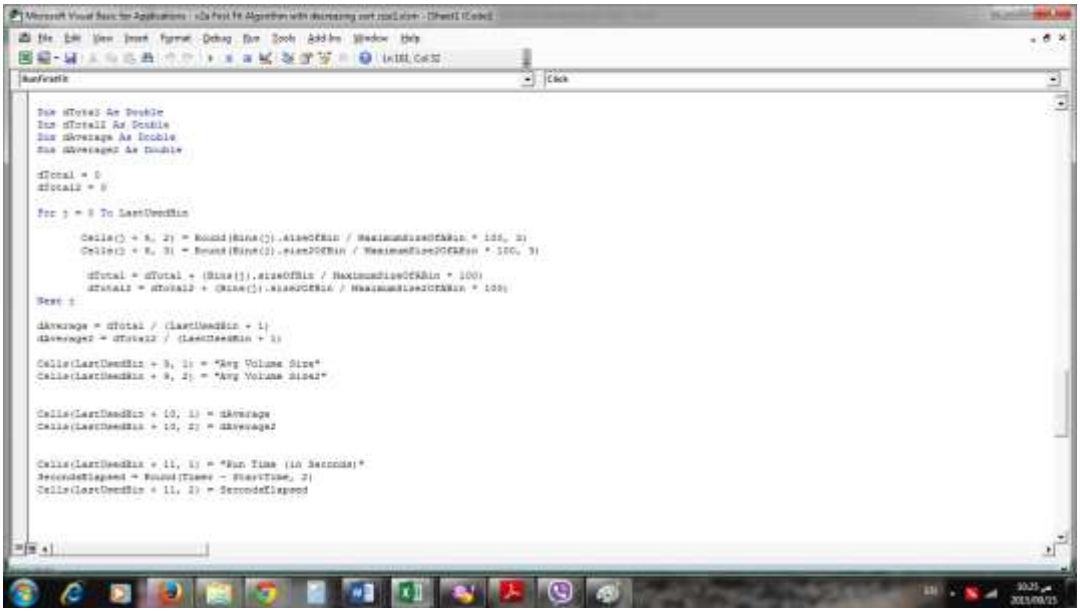
```
Dim dTotal As Double
Dim dTotal2 As Double
Dim dAverage As Double
Dim dAverage2 As Double

dTotal = 0
dTotal2 = 0

For j = 0 To LastUsedBin

       Cells(j + 8, 2) = Round(Bins(j).sizeOfBin / MaximumSizeOfABin * 100, 3)
       Cells(j + 8, 3) = Round(Bins(j).size2OfBin / MaximumSize2OfABin * 100, 3)

        dTotal = dTotal + (Bins(j).sizeOfBin / MaximumSizeOfABin * 100)
        dTotal2 = dTotal2 + (Bins(j).size2OfBin / MaximumSize2OfABin * 100)
Next j

dAverage = dTotal / (LastUsedBin + 1)
dAverage2 = dTotal2 / (LastUsedBin + 1)

Cells(LastUsedBin + 9, 1) = "Avg Volume Size"
Cells(LastUsedBin + 9, 2) = "Avg Volume Size2"


Cells(LastUsedBin + 10, 1) = dAverage
Cells(LastUsedBin + 10, 2) = dAverage2


Cells(LastUsedBin + 11, 1) = "Run Time (in Seconds)"
SecondsElapsed = Round(Timer - StartTime, 2)
Cells(LastUsedBin + 11, 2) = SecondsElapsed

End Sub
```

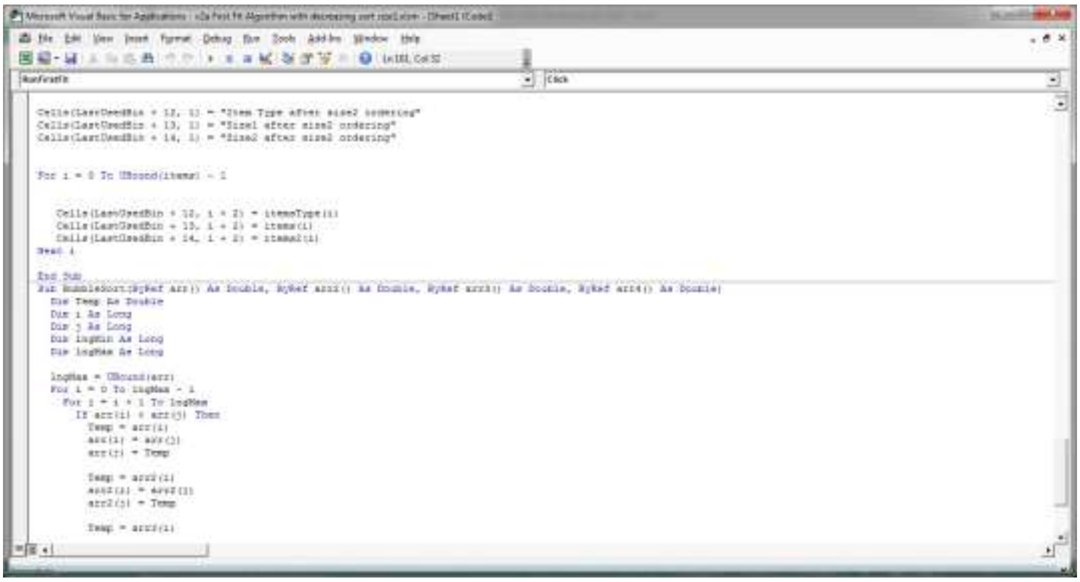
```
Sub BubbleSort(ByRef arr() As Double, ByRef arr2() As Double, ByRef arr3() As Double)
  Dim Temp As Double
  Dim i As Long
  Dim j As Long
  Dim lngMin As Long
  Dim lngMax As Long

  lngMax = UBound(arr)
  For i = 0 To lngMax - 1
    For j = i + 1 To lngMax
      If arr(i) < arr(j) Then
        Temp = arr(i)
        arr(i) = arr(j)
        arr(j) = Temp

        Temp = arr2(i)
        arr2(i) = arr2(j)
        arr2(j) = Temp

        Temp = arr3(i)
        arr3(i) = arr3(j)
        arr3(j) = Temp
      End If
    Next j
  Next i
End Sub
```

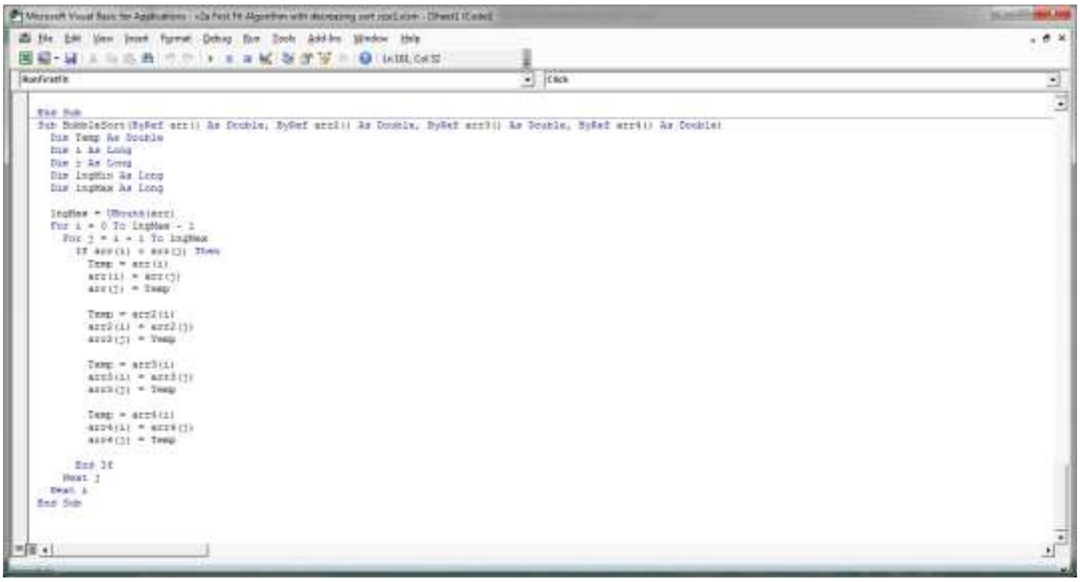
This is the configuration code of the algorithm 2






Dim SumDemands As Double
SumDemands = 0
For i = 0 To UBound(items) - 1
SumDemands = SumDemands + demand(i)
Next i
ReDim Bins(SumDemands) As clsBin
For i = 0 To SumDemands - 1
Set Bins(i) = New clsBin
Bins(i).updateLastUsedPlace = 1
Bins(i).updateSize = 0
Bins(i).updateSize2 = 0
Next i
Dim LastUsedBin As Double
LastUsedBin = 0
For i = 0 To UBound(items) - 1
foundAplaceForTheItem = False
For j = 0 To LastUsedBin
If (Bins(j).sizeOfBin + items(i) <= MaximumSizeOfABin And Bins(j).size2OfBin + items2(i) <= MaximumSize2OfABin) Then
Bins(j).updateSize = Bins(j).sizeOfBin + items(i)

For j = 0 To LastUsedBin
If (Bins(j).sizeOfBin + items(i) <= MaximumSizeOfABin And Bins(j).size2OfBin + items2(i) <= MaximumSize2OfABin) Then
Bins(j).updateSize = Bins(j).sizeOfBin + items(i)
Bins(j).updateSize2 = Bins(j).size2OfBin + items2(i)
Bins(j).updateLastUsedPlace = Bins(j).LastUsedPlaceOfBin + 1
Cells(j + 8, Bins(j).LastUsedPlaceOfBin + 1) = itemsType(i)
foundAplaceForTheItem = True
Exit For
End If
Next j
If (Not foundAplaceForTheItem) Then
LastUsedBin = LastUsedBin + 1
Cells(LastUsedBin + 8, 1) = "Bin " & CStr(LastUsedBin + 1)
Bins(LastUsedBin).updateSize = Bins(LastUsedBin).sizeOfBin + items(i)
Bins(LastUsedBin).updateSize2 = Bins(LastUsedBin).size2OfBin + items2(i)
Bins(LastUsedBin).updateLastUsedPlace = Bins(LastUsedBin).LastUsedPlaceOfBin + 1
Cells(LastUsedBin + 8, Bins(LastUsedBin).LastUsedPlaceOfBin + 1) = itemsType(i)
foundAplaceForTheItem = True
End If
demand(i) = demand(i) - 1
If (demand(i) > 0) Then
i = i - 1
End If
Next i

Dim dTotal As Double
Dim dTotal2 As Double
Dim dAverage As Double
Dim dAverage2 As Double
dTotal = 0
dTotal2 = 0
For j = 0 To LastUsedBin
Cells(j + 8, 2) = Round(Bins(j).sizeOfBin / MaximumSizeOfABin * 100, 2)
Cells(j + 8, 3) = Round(Bins(j).size2OfBin / MaximumSize2OfABin * 100, 3)
dTotal = dTotal + (Bins(j).sizeOfBin / MaximumSizeOfABin * 100)
dTotal2 = dTotal2 + (Bins(j).size2OfBin / MaximumSize2OfABin * 100)
Next j
dAverage = dTotal / (LastUsedBin + 1)
dAverage2 = dTotal2 / (LastUsedBin + 1)
Cells(LastUsedBin + 9, 1) = "Avg Volume Size"
Cells(LastUsedBin + 9, 2) = "Avg Volume Size2"
Cells(LastUsedBin + 10, 1) = dAverage
Cells(LastUsedBin + 10, 2) = dAverage2
Cells(LastUsedBin + 11, 1) = "Run Time (in Seconds)"
SecondsElapsed = Round(Timer - StartTime, 2)
Cells(LastUsedBin + 11, 2) = SecondsElapsed


Cells(LastUsedBin + 12, 1) = "Item Type after size2 ordering"
Cells(LastUsedBin + 13, 1) = "Size1 after size2 ordering"
Cells(LastUsedBin + 14, 1) = "Size2 after size2 ordering"
For i = 0 To UBound(items) - 1
Cells(LastUsedBin + 12, i + 2) = itemsType(i)
Cells(LastUsedBin + 13, i + 2) = items(i)
Cells(LastUsedBin + 14, i + 2) = items2(i)
Next i
End Sub
Sub BubbleSort(ByRef arr() As Double, ByRef arr2() As Double, ByRef arr3() As Double, ByRef arr4() As Double)
Dim Temp As Double
Dim i As Long
Dim j As Long
Dim lngMin As Long
Dim lngMax As Long
lngMax = UBound(arr)
For i = 0 To lngMax - 1
For j = i + 1 To lngMax
Temp = arr(i)
arr(i) = arr(j)
arr(j) = Temp
Temp = arr2(i)
arr2(i) = arr2(j)
arr2(j) = Temp

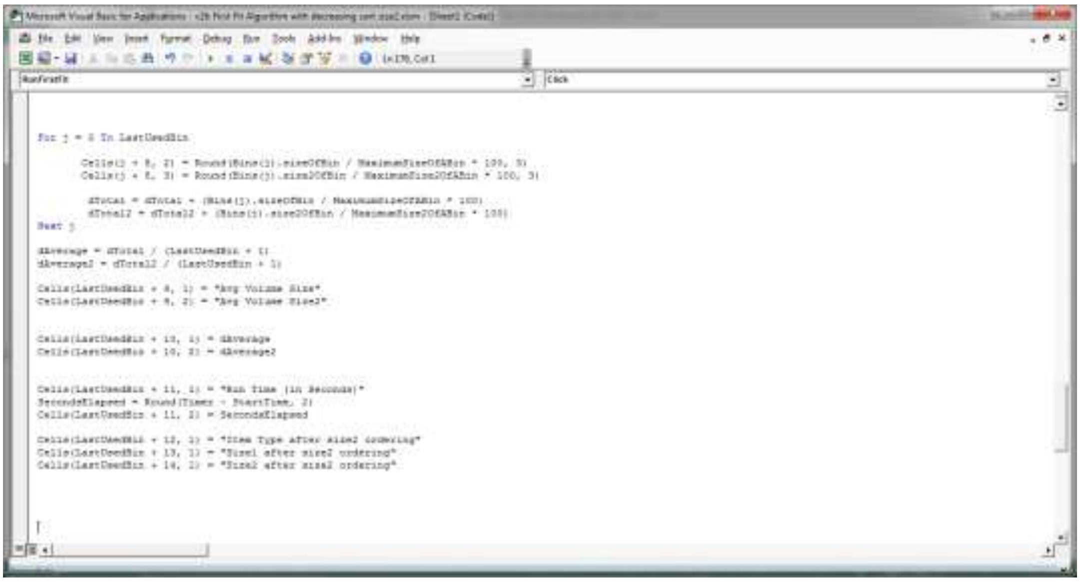

End Sub
Sub BubbleSort(ByRef arr() As Double, ByRef arr2() As Double, ByRef arr3() As Double, ByRef arr4() As Double)
Dim Temp As Double
Dim i As Long
Dim j As Long
Dim lngMin As Long
Dim lngMax As Long
lngMax = UBound(arr)
For i = 0 To lngMax - 1
For j = i + 1 To lngMax
Temp = arr(i)
arr(i) = arr(j)
arr(j) = Temp
Temp = arr2(i)
arr2(i) = arr2(j)
arr2(j) = Temp
Temp = arr3(i)
arr3(i) = arr3(j)
arr3(j) = Temp
Temp = arr4(i)
arr4(i) = arr4(j)
arr4(j) = Temp
End If
Next j
Next i
End Sub

This is the configuration code of the algorithm 3

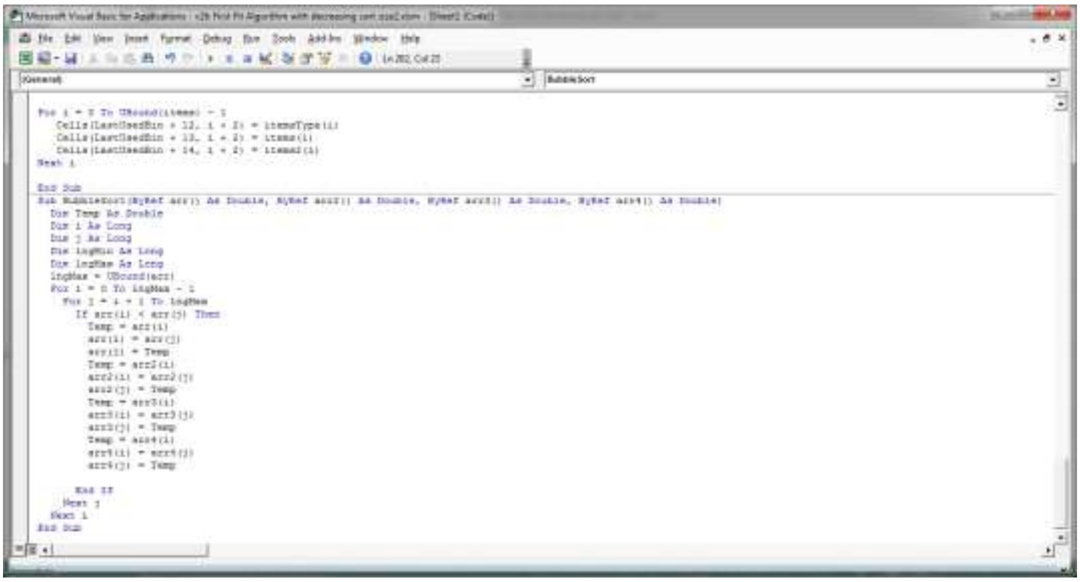
```vb
ReDim itemsType(NoOfItems)
ReDim items(NoOfItems)
ReDim items2(NoOfItems)
ReDim demand(NoOfItems)

Dim foundAplaceForTheItem As Boolean

Cells(8, 1) = "Bin 1"
Cells(7, 2) = "Volume Size 1 Perc%"
Cells(7, 3) = "Volume Size 2 Perc%"

For i = 0 To UBound(items) - 1
  itemsType(i) = Cells(1, i + 2)
  items(i) = Cells(2, i + 2)

  items2(i) = Cells(3, i + 2)
  demand(i) = Cells(4, i + 2)

Next i


 Call BubbleSort(items2, items, demand, itemsType)



Dim SumDemands As Double

SumDemands = 0

For i = 0 To UBound(items) - 1

  SumDemands = SumDemands + demand(i)

Next i
```

```vb
ReDim Bins(SumDemands) As clsBin


For i = 0 To SumDemands - 1
   Set Bins(i) = New clsBin
   Bins(i).updateLastUsedPlace = 2
   Bins(i).updateSize = 0
   Bins(i).updateSize2 = 0

Next i

Dim LastUsedBin As Double


LastUsedBin = 0
For i = 0 To UBound(items) - 1

  foundAplaceForTheItem = False


  For j = 0 To LastUsedBin
    If (Bins(j).sizeOfBin + items(i) <= MaximumSizeOfABin And Bins(j).size2OfBin + items2(i) <= MaximumSize2OfABin) Then

        Bins(j).updateSize = Bins(j).sizeOfBin + items(i)
        Bins(j).updateSize2 = Bins(j).size2OfBin + items2(i)
        Bins(j).updateLastUsedPlace = Bins(j).LastUsedPlaceOfBin + 1
        Cells(j + 8, Bins(j).LastUsedPlaceOfBin + 1) = itemsType(i)
        foundAplaceForTheItem = True
        Exit For
    End If

  Next j
```

```vb
  If (Not foundAplaceForTheItem) Then
    LastUsedBin = LastUsedBin + 1

    Cells(LastUsedBin + 8, 1) = "Bin " & CStr(LastUsedBin + 1)

    Bins(LastUsedBin).updateSize = Bins(LastUsedBin).sizeOfBin + items(i)
    Bins(LastUsedBin).updateSize2 = Bins(LastUsedBin).size2OfBin + items2(i)
    Bins(LastUsedBin).updateLastUsedPlace = Bins(LastUsedBin).LastUsedPlaceOfBin + 1

    Cells(LastUsedBin + 8, Bins(LastUsedBin).LastUsedPlaceOfBin + 1) = itemsType(i)
    foundAplaceForTheItem = True


  End If
  ' the following is necessary in case demand will be count and handled whenever met (in randomizing vector)
  demand(i) = demand(i) - 1
  If (demand(i) > 0) Then
   i = i - 1
  End If

Next i

Dim dTotal As Double
Dim dTotal2 As Double
Dim dAverage As Double
Dim dAverage2 As Double

dTotal = 0
dTotal2 = 0
```

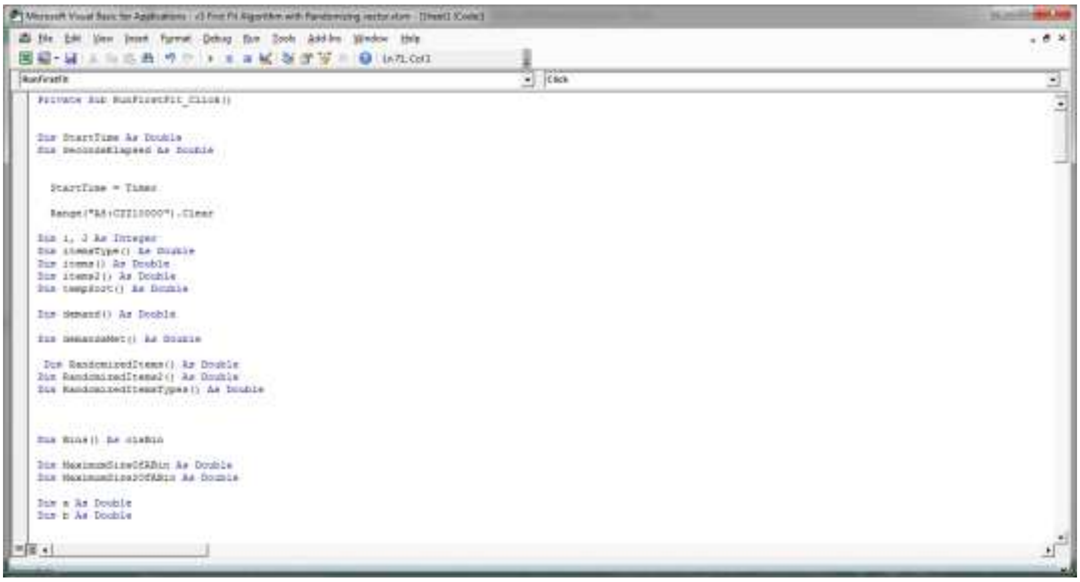


```
For j = 0 To LastUsedBin

        Cells(j + 8, 2) = Round(Bins(j).sizeOfBin / MaximumSizeOfABin * 100, 3)
        Cells(j + 8, 3) = Round(Bins(j).size2OfBin / MaximumSize2OfABin * 100, 3)

         dTotal = dTotal + (Bins(j).sizeOfBin / MaximumSizeOfABin * 100)
         dTotal2 = dTotal2 + (Bins(j).size2OfBin / MaximumSize2OfABin * 100)
Next j

dAverage = dTotal / (LastUsedBin + 1)
dAverage2 = dTotal2 / (LastUsedBin + 1)

Cells(LastUsedBin + 9, 1) = "Avg Volume Size"
Cells(LastUsedBin + 9, 2) = "Avg Volume Size2"


Cells(LastUsedBin + 10, 1) = dAverage
Cells(LastUsedBin + 10, 2) = dAverage2


Cells(LastUsedBin + 11, 1) = "Run Time (in Seconds)"
SecondsElapsed = Round(Timer - StartTime, 2)
Cells(LastUsedBin + 11, 2) = SecondsElapsed

Cells(LastUsedBin + 12, 1) = "Item Type after size2 ordering"
Cells(LastUsedBin + 13, 1) = "Size1 after size2 ordering"
Cells(LastUsedBin + 14, 1) = "Size2 after size2 ordering"
```

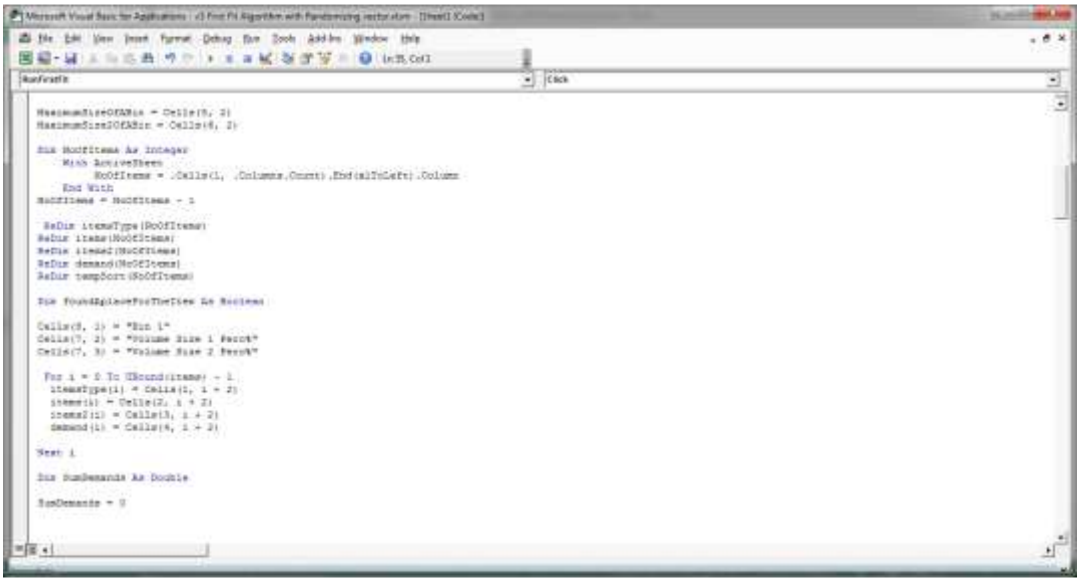


```
For i = 0 To UBound(items) - 1
   Cells(LastUsedBin + 12, i + 2) = itemsType(i)
   Cells(LastUsedBin + 13, i + 2) = items(i)
   Cells(LastUsedBin + 14, i + 2) = items2(i)
Next i

End Sub
Sub BubbleSort(ByRef arr() As Double, ByRef arr2() As Double, ByRef arr3() As Double, ByRef arr4() As Double)
  Dim Temp As Double
  Dim i As Long
  Dim j As Long
  Dim lngMin As Long
  Dim lngMax As Long
  lngMax = UBound(arr)
  For i = 0 To lngMax - 1
    For j = i + 1 To lngMax
      If arr(i) < arr(j) Then
        Temp = arr(i)
        arr(i) = arr(j)
        arr(j) = Temp
        Temp = arr2(i)
        arr2(i) = arr2(j)
        arr2(j) = Temp
        Temp = arr3(i)
        arr3(i) = arr3(j)
        arr3(j) = Temp
        Temp = arr4(i)
        arr4(i) = arr4(j)
        arr4(j) = Temp

      End If
    Next j
  Next i
End Sub
```

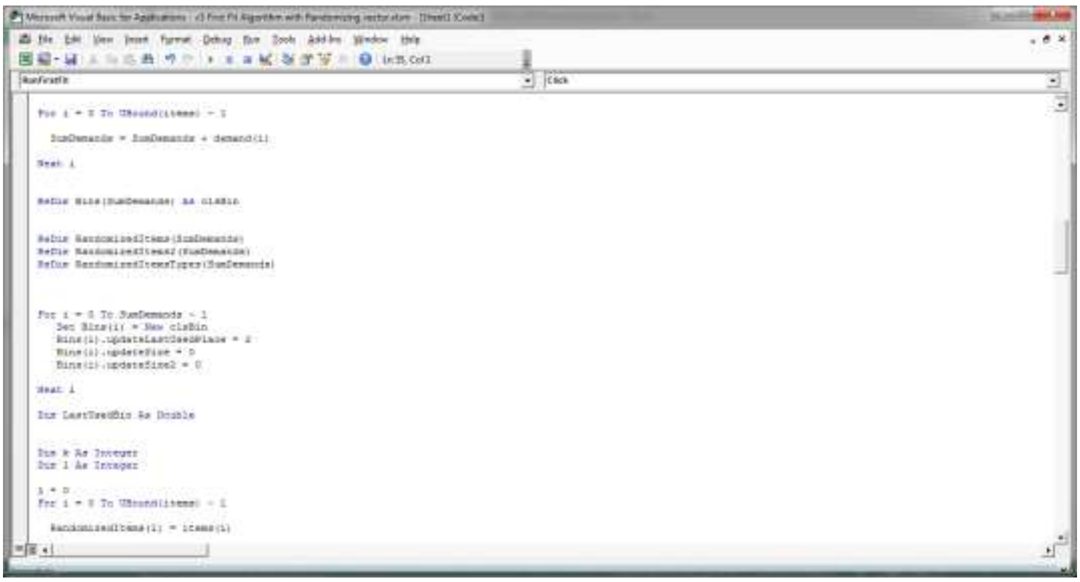
This is the configuration code of the algorithm 4



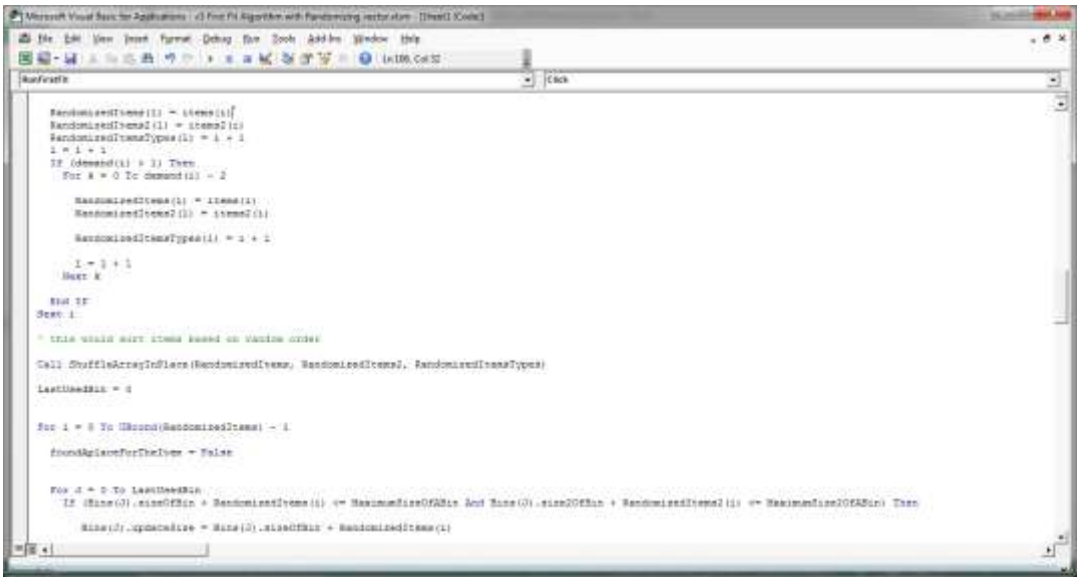
```
Private Sub RunFirstFit_Click()


Dim StartTime As Double
Dim SecondsElapsed As Double


  StartTime = Timer

  Range("A8:CZZ10000").Clear

Dim i, j As Integer
Dim itemsType() As Double
Dim items() As Double
Dim items2() As Double
Dim tempSort() As Double

Dim demand() As Double

Dim demandsMet() As Double

 Dim RandomizedItems() As Double
Dim RandomizedItems2() As Double
Dim RandomizedItemsTypes() As Double



Dim Bins() As clsBin

Dim MaximumSizeOfABin As Double
Dim MaximumSize2OfABin As Double

Dim a As Double
Dim b As Double
```



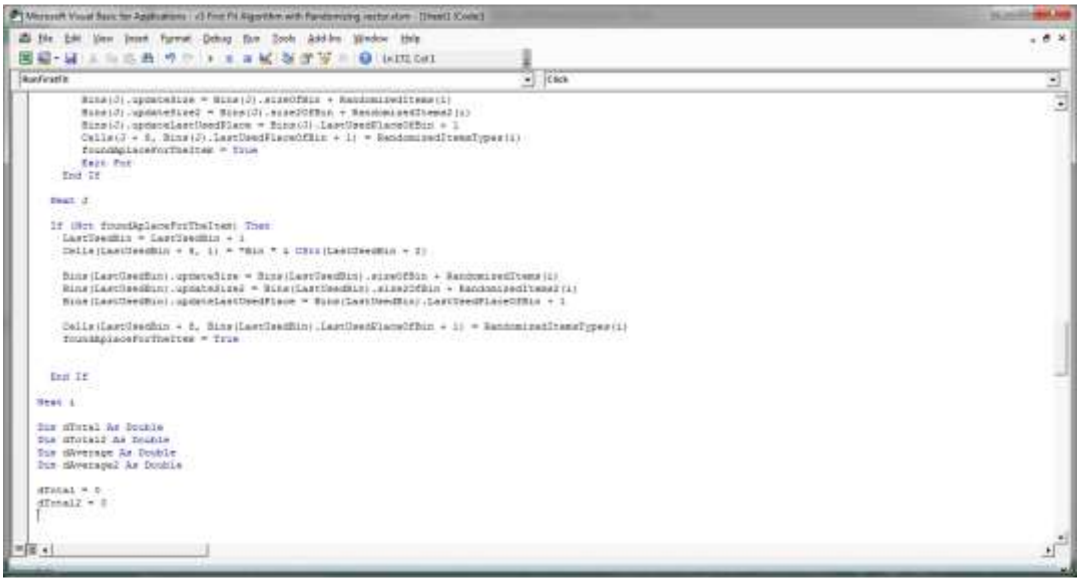
```
MaximumSizeOfABin = Cells(5, 2)
MaximumSize2OfABin = Cells(6, 2)

Dim NoOfItems As Integer
    With ActiveSheet
        NoOfItems = .Cells(1, .Columns.Count).End(xlToLeft).Column
    End With
NoOfItems = NoOfItems - 1

 ReDim itemsType(NoOfItems)
ReDim items(NoOfItems)
ReDim items2(NoOfItems)
ReDim demand(NoOfItems)
ReDim tempSort(NoOfItems)

Dim FoundAplaceForTheItem As Boolean

Cells(8, 1) = "Bin 1"
Cells(7, 2) = "Volume Size 1 Perct"
Cells(7, 3) = "Volume Size 2 Perct"

 For i = 0 To UBound(items) - 1
  itemsType(i) = Cells(1, i + 2)
  items(i) = Cells(2, i + 2)
  items2(i) = Cells(3, i + 2)
  demand(i) = Cells(4, i + 2)

Next i

Dim SumDemands As Double

SumDemands = 0
```



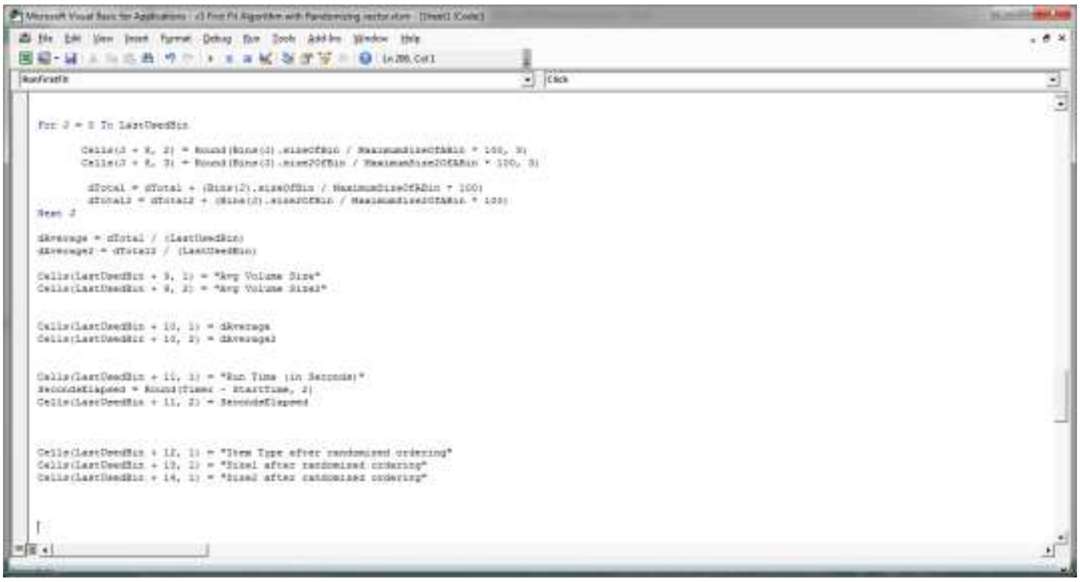
```
For i = 0 To UBound(items) - 1

  SumDemands = SumDemands + demand(i)

Next i


ReDim Bins(SumDemands) As clsBin


ReDim RandomizedItems(SumDemands)
ReDim RandomizedItems2(SumDemands)
ReDim RandomizedItemsTypes(SumDemands)



For i = 0 To SumDemands - 1
   Set Bins(i) = New clsBin
   Bins(i).updateLastUsedPlace = 2
   Bins(i).updateSize = 0
   Bins(i).updateSize2 = 0

Next i

Dim LastUsedBin As Double


Dim k As Integer
Dim l As Integer

l = 0
For i = 0 To UBound(items) - 1

  RandomizedItems(l) = items(i)
```



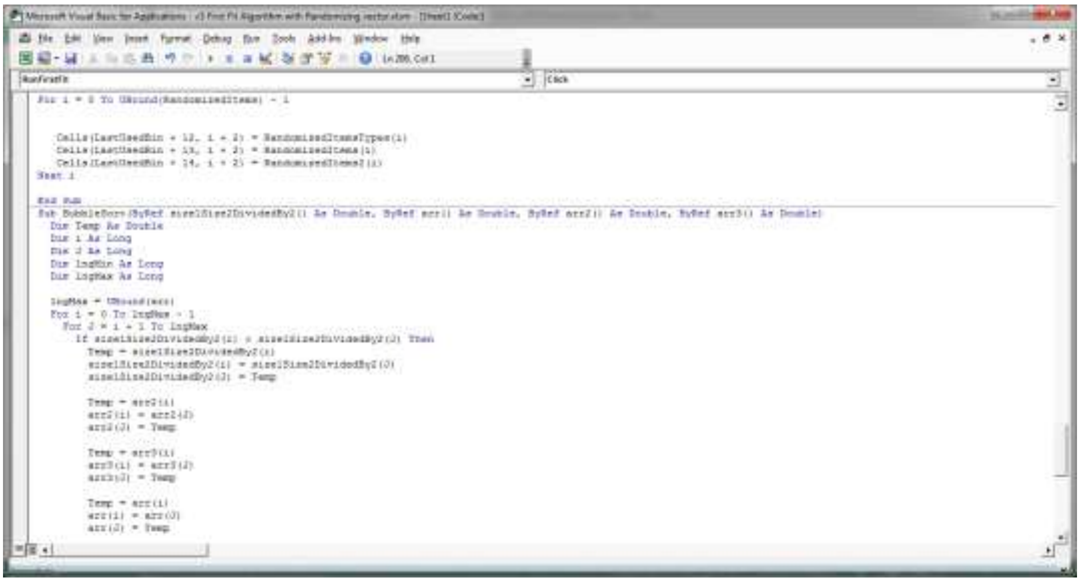
```
    RandomizedItems(i) = items(i)
    RandomizedItems2(i) = items2(i)
    RandomizedItemsTypes(i) = i + 1
    i = i + 1
    If (demand(i) > 1) Then
      For k = 0 To demand(i) - 2

        RandomizedItems(i) = items(i)
        RandomizedItems2(i) = items2(i)

        RandomizedItemsTypes(i) = i + 1

        I = i + 1
      Next k

    End If
  Next i

  ' this would sort items based on random order

  Call ShuffleArrayInPlace(RandomizedItems, RandomizedItems2, RandomizedItemsTypes)

  LastUsedBin = 0


  For i = 0 To UBound(RandomizedItems) - 1

    foundAplaceForTheItem = False


    For J = 0 To LastUsedBin
      If (Bins(J).sizeOfBin + RandomizedItems(i) <= MaximumSizeOfABin And Bins(J).size2OfBin + RandomizedItems2(i) <= MaximumSize2OfABin) Then

        Bins(J).updateSize = Bins(J).sizeOfBin + RandomizedItems(i)
```

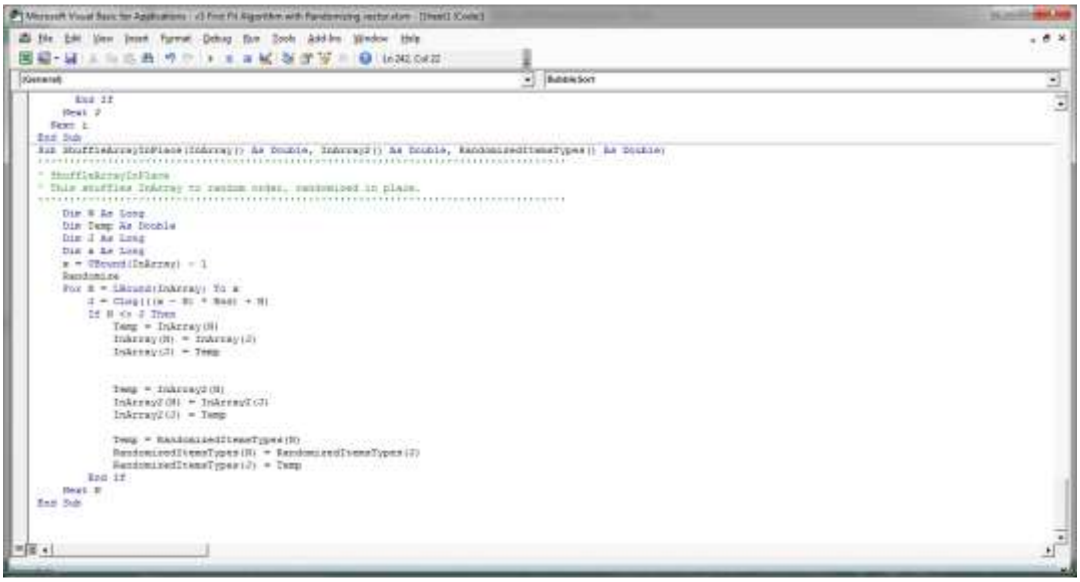


```
        Bins(J).updateSize = Bins(J).sizeOfBin + RandomizedItems(i)
        Bins(J).updateSize2 = Bins(J).size2OfBin + RandomizedItems2(i)
        Bins(J).updateLastUsedPlace = Bins(J).LastUsedPlaceOfBin + 1
        Cells(J + 8, Bins(J).LastUsedPlaceOfBin + 1) = RandomizedItemsTypes(i)
        foundAplaceForTheItem = True
        Exit For
      End If

    Next J

    If (Not foundAplaceForTheItem) Then
      LastUsedBin = LastUsedBin + 1
      Cells(LastUsedBin + 8, 1) = "Bin " & CStr(LastUsedBin + 1)

      Bins(LastUsedBin).updateSize = Bins(LastUsedBin).sizeOfBin + RandomizedItems(i)
      Bins(LastUsedBin).updateSize2 = Bins(LastUsedBin).size2OfBin + RandomizedItems2(i)
      Bins(LastUsedBin).updateLastUsedPlace = Bins(LastUsedBin).LastUsedPlaceOfBin + 1

      Cells(LastUsedBin + 8, Bins(LastUsedBin).LastUsedPlaceOfBin + 1) = RandomizedItemsTypes(i)
      foundAplaceForTheItem = True


    End If

  Next i

  Dim dTotal As Double
  Dim dTotal2 As Double
  Dim dAverage As Double
  Dim dAverage2 As Double

  dTotal = 0
  dTotal2 = 0
```

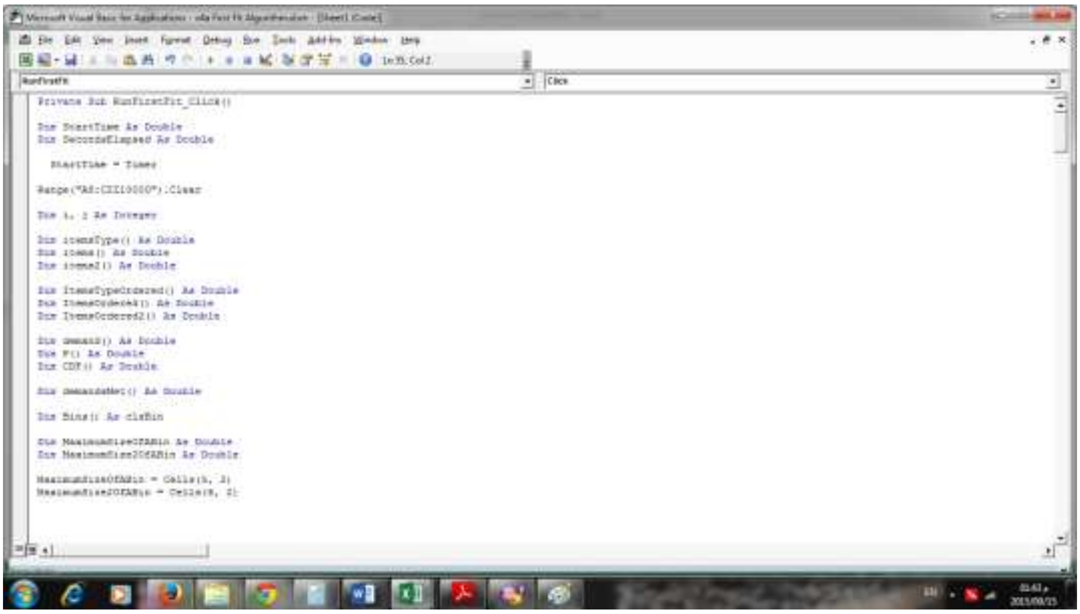


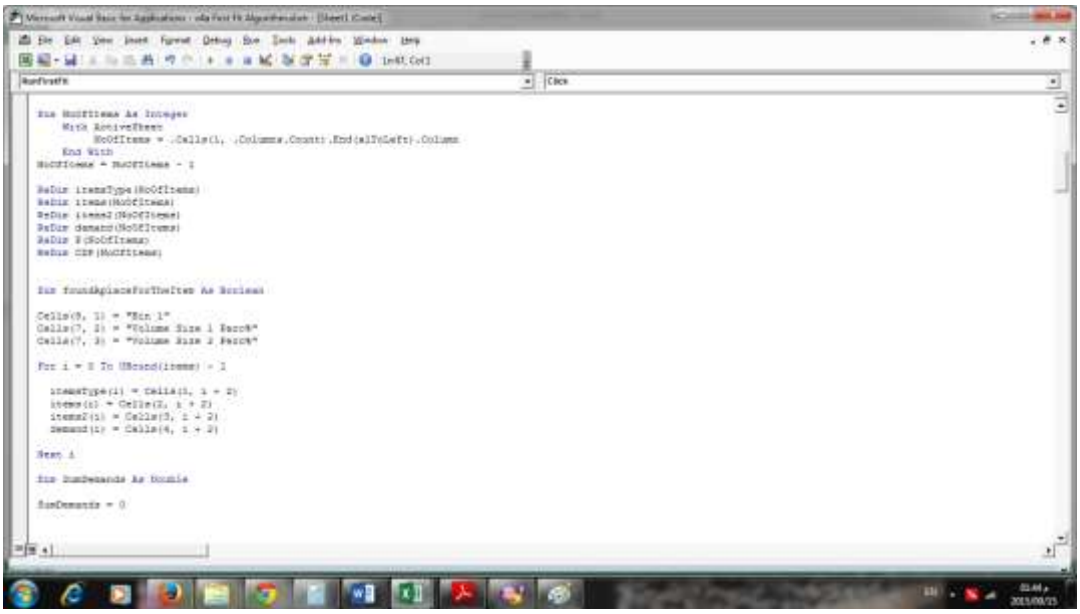
```
  For J = 0 To LastUsedBin

        Cells(J + 8, 2) = Round(Bins(J).sizeOfBin / MaximumSizeOfABin * 100, 3)
        Cells(J + 8, 3) = Round(Bins(J).size2OfBin / MaximumSize2OfABin * 100, 3)

         dTotal = dTotal + (Bins(J).sizeOfBin / MaximumSizeOfABin * 100)
         dTotal2 = dTotal2 + (Bins(J).size2OfBin / MaximumSize2OfABin * 100)
  Next J

  dAverage = dTotal / (LastUsedBin)
  dAverage2 = dTotal2 / (LastUsedBin)

  Cells(LastUsedBin + 9, 1) = "Avg Volume Size"
  Cells(LastUsedBin + 9, 2) = "Avg Volume Size2"


  Cells(LastUsedBin + 10, 1) = dAverage
  Cells(LastUsedBin + 10, 2) = dAverage2


  Cells(LastUsedBin + 11, 1) = "Run Time (in Seconds)"
  SecondsElapsed = Round(Timer - StartTime, 2)
  Cells(LastUsedBin + 11, 2) = SecondsElapsed



  Cells(LastUsedBin + 12, 1) = "Item Type after randomized ordering"
  Cells(LastUsedBin + 13, 1) = "Size1 after randomized ordering"
  Cells(LastUsedBin + 14, 1) = "Size2 after randomized ordering"
```

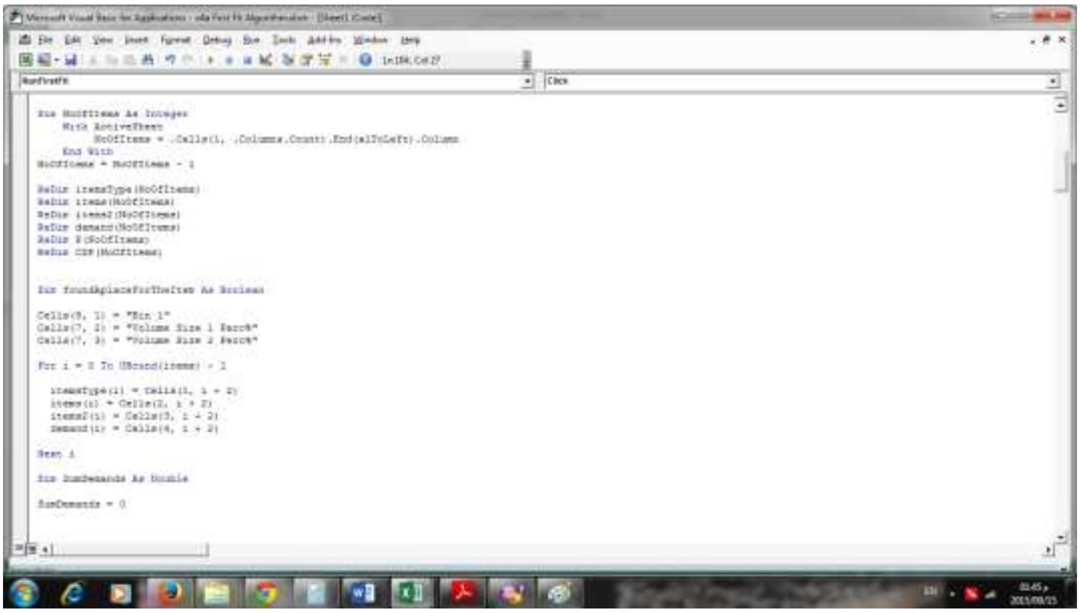

Microsoft Visual Basic for Applications - v3 First Fit Algorithm with Randomizing vector.xlsm - [Sheet1 (Code)]
File Edit View Insert Format Debug Run Tools Add-Ins Window Help
Ln 206, Col 1
RunFirstFit
Click
For i = 0 To UBound(RandomizedItems) - 1
Cells(LastUsedBin + 12, i + 2) = RandomizedItemsTypes(i)
Cells(LastUsedBin + 13, i + 2) = RandomizedItems(i)
Cells(LastUsedBin + 14, i + 2) = RandomizedItems2(i)
Next i
End Sub
Sub BubbleSort(ByRef size1Size2DividedBy2() As Double, ByRef arr() As Double, ByRef arr2() As Double, ByRef arr3() As Double)
Dim Temp As Double
Dim i As Long
Dim J As Long
Dim lngMin As Long
Dim lngMax As Long
lngMax = UBound(arr)
For i = 0 To lngMax - 1
For J = i + 1 To lngMax
If size1Size2DividedBy2(i) > size1Size2DividedBy2(J) Then
Temp = size1Size2DividedBy2(i)
size1Size2DividedBy2(i) = size1Size2DividedBy2(J)
size1Size2DividedBy2(J) = Temp
Temp = arr2(i)
arr2(i) = arr2(J)
arr2(J) = Temp
Temp = arr3(i)
arr3(i) = arr3(J)
arr3(J) = Temp
Temp = arr(i)
arr(i) = arr(J)
arr(J) = Temp


Microsoft Visual Basic for Applications - v3 First Fit Algorithm with Randomizing vector.xlsm - [Sheet1 (Code)]
File Edit View Insert Format Debug Run Tools Add-Ins Window Help
Ln 242, Col 22
(General)
BubbleSort
End If
Next J
Next i
End Sub
Sub ShuffleArrayInPlace(InArray() As Double, InArray2() As Double, RandomizedItemsTypes() As Double)
' ShuffleArrayInPlace
' This shuffles InArray to random order, randomized in place.
Dim N As Long
Dim Temp As Double
Dim J As Long
Dim a As Long
a = UBound(InArray) - 1
Randomize
For N = LBound(InArray) To a
J = CLng(((a - N) * Rnd) + N)
If N <> J Then
Temp = InArray(N)
InArray(N) = InArray(J)
InArray(J) = Temp
Temp = InArray2(N)
InArray2(N) = InArray2(J)
InArray2(J) = Temp
Temp = RandomizedItemsTypes(N)
RandomizedItemsTypes(N) = RandomizedItemsTypes(J)
RandomizedItemsTypes(J) = Temp
End If
Next N
End Sub

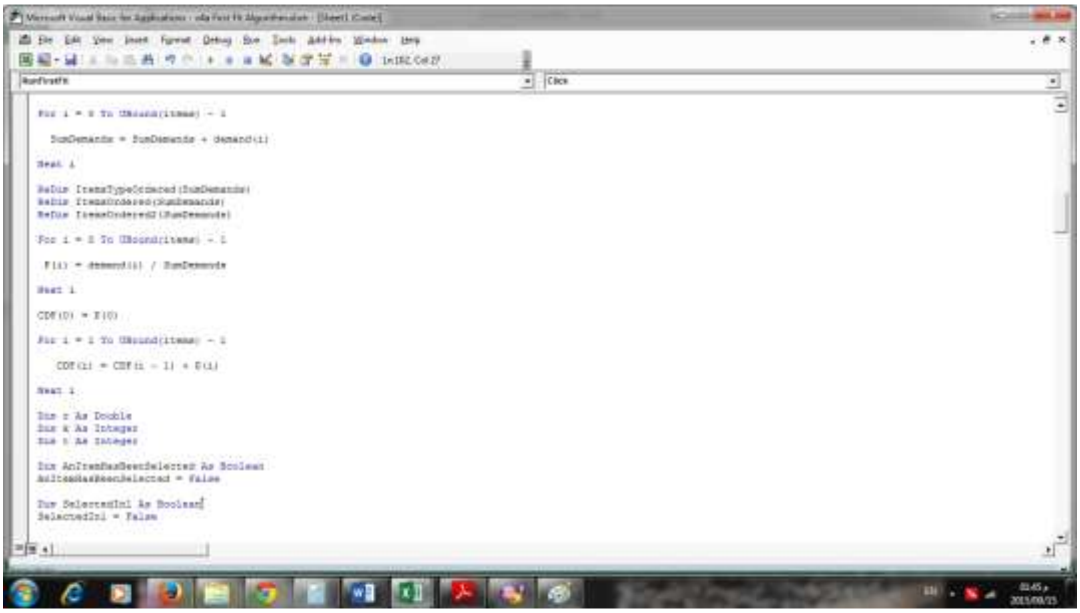
This is the configuration code of the algorithm 5



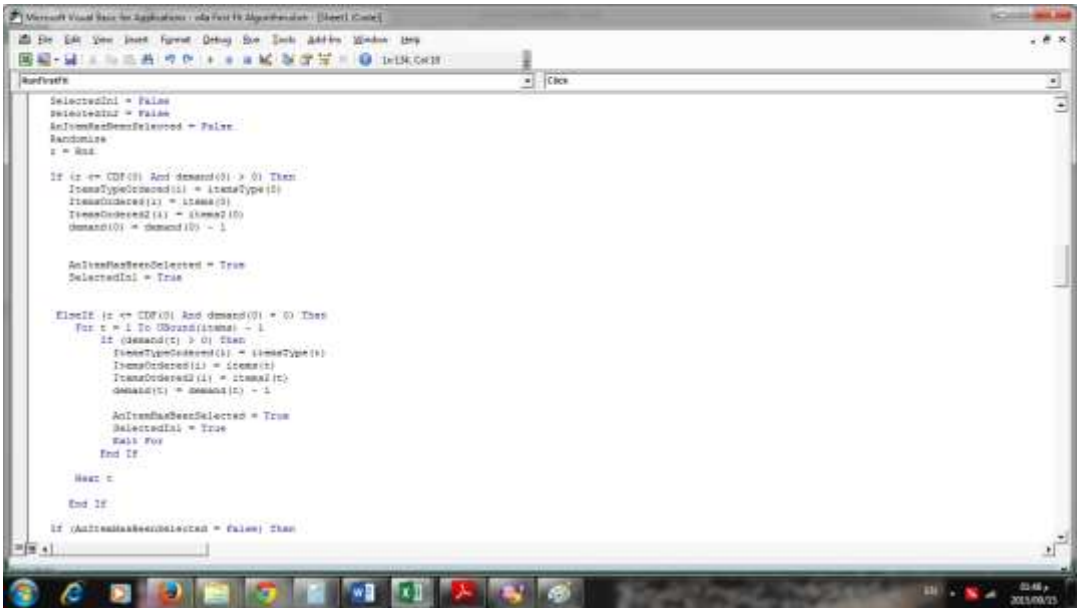




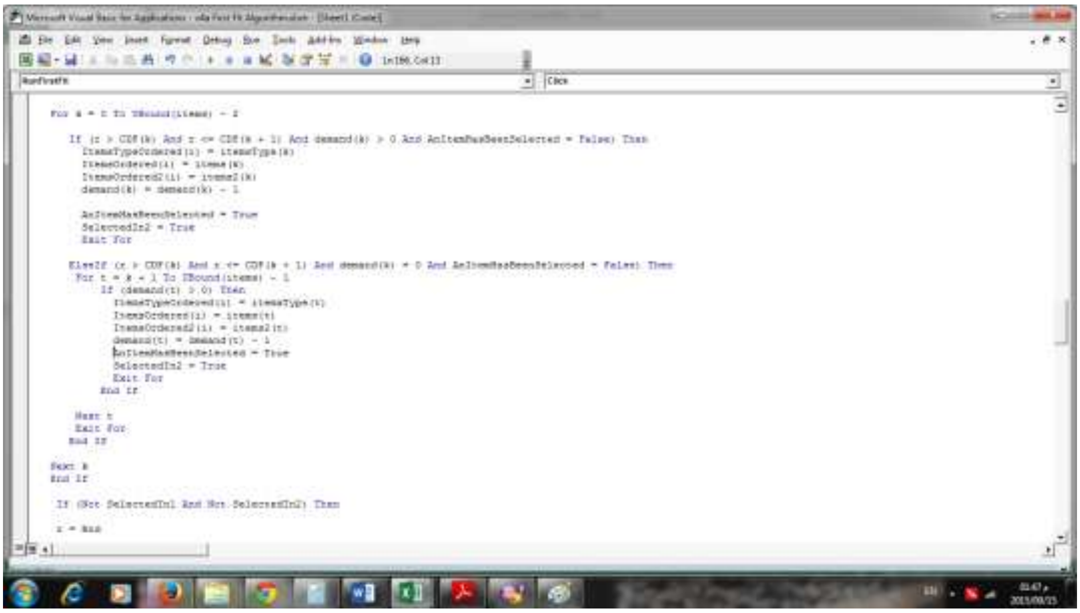
```
Dim NoOfItems As Integer
    With ActiveSheet
        NoOfItems = .Cells(1, .Columns.Count).End(xlToLeft).Column
    End With
NoOfItems = NoOfItems - 1

ReDim itemsType(NoOfItems)
ReDim items(NoOfItems)
ReDim items2(NoOfItems)
ReDim demand(NoOfItems)
ReDim F(NoOfItems)
ReDim CDF(NoOfItems)


Dim foundAplaceForTheItem As Boolean

Cells(9, 1) = "Bin 1"
Cells(7, 2) = "Volume Size 1 Perc%"
Cells(7, 3) = "Volume Size 2 Perc%"

For i = 1 To UBound(items) - 1

  itemsType(i) = Cells(1, i + 2)
  items(i) = Cells(2, i + 2)
  items2(i) = Cells(3, i + 2)
  demand(i) = Cells(4, i + 2)

Next i

Dim SumDemands As Double

SumDemands = 0
```



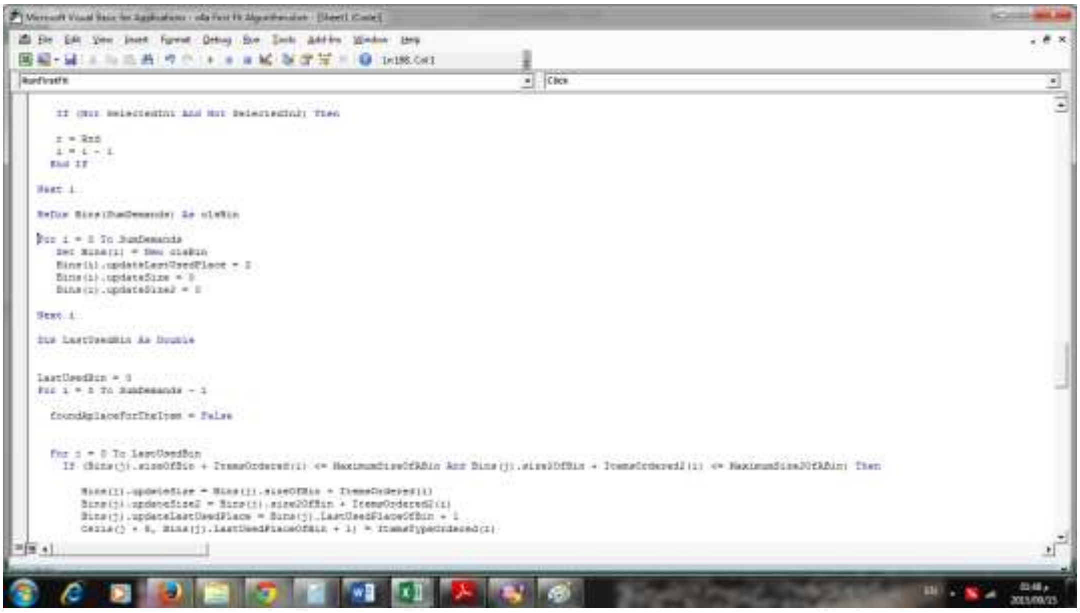
```
For i = 1 To UBound(items) - 1

  SumDemands = SumDemands + demand(i)

Next i

ReDim ItemsTypeOrdered(SumDemands)
ReDim ItemsOrdered(SumDemands)
ReDim ItemsOrdered2(SumDemands)

For i = 1 To UBound(items) - 1

 F(i) = demand(i) / SumDemands

Next i

CDF(0) = F(0)

For i = 1 To UBound(items) - 1

   CDF(i) = CDF(i - 1) + F(i)

Next i

Dim r As Double
Dim k As Integer
Dim t As Integer

Dim AnItemHasBeenSelected As Boolean
AnItemHasBeenSelected = False

Dim SelectedIn1 As Boolean
SelectedIn1 = False
```



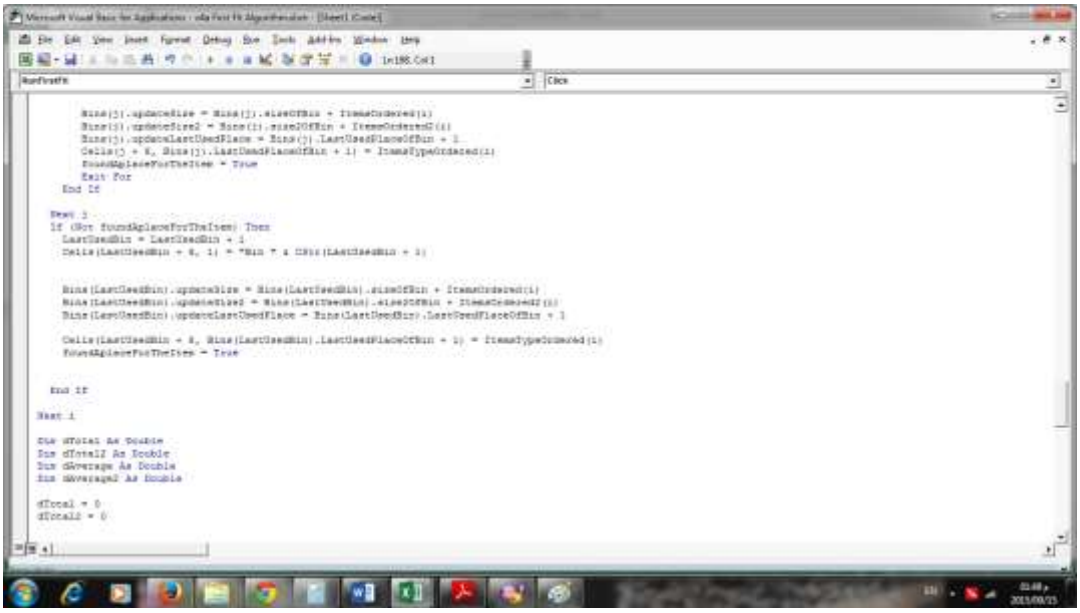
```
  SelectedIn1 = False
  SelectedIn2 = False
  AnItemHasBeenSelected = False
  Randomize
  r = Rnd

  If (r <= CDF(0) And demand(0) > 0) Then
     ItemsTypeOrdered(i) = itemsType(0)
     ItemsOrdered(i) = items(0)
     ItemsOrdered2(i) = items2(0)
     demand(0) = demand(0) - 1


     AnItemHasBeenSelected = True
     SelectedIn1 = True


   ElseIf (r <= CDF(0) And demand(0) = 0) Then
      For t = 1 To UBound(items) - 1
          If (demand(t) > 0) Then
            ItemsTypeOrdered(i) = itemsType(t)
            ItemsOrdered(i) = items(t)
            ItemsOrdered2(i) = items2(t)
            demand(t) = demand(t) - 1

            AnItemHasBeenSelected = True
            SelectedIn1 = True
            Exit For
          End If

      Next t

     End If

  If (AnItemHasBeenSelected = False) Then
```



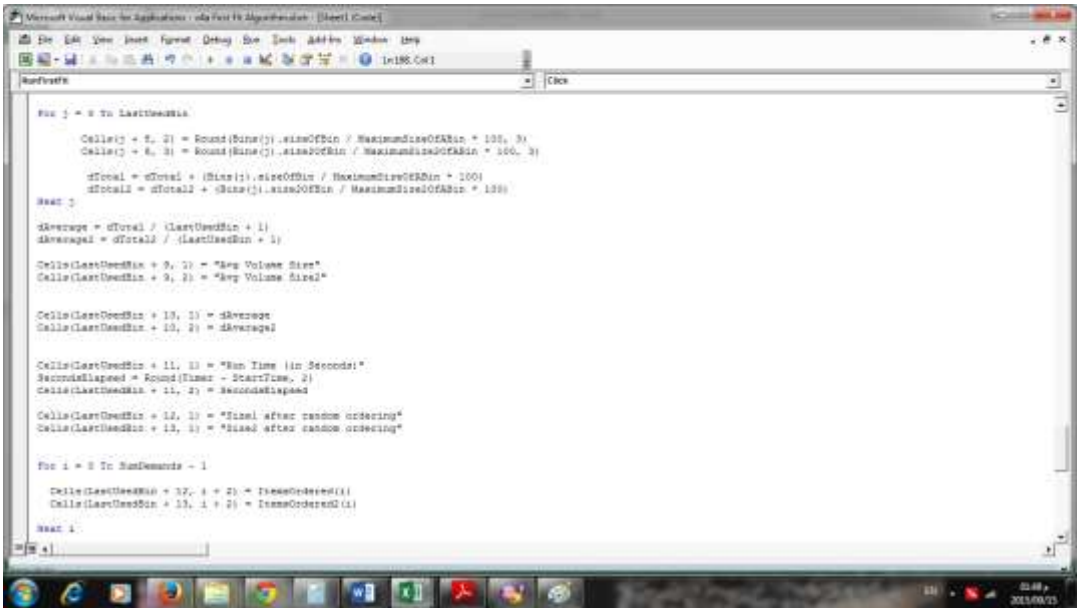
```
For k = 0 To UBound(items) - 1

   If (r > CDF(k) And r <= CDF(k + 1) And demand(k) > 0 And AnItemHasBeenSelected = False) Then
     ItemsTypeOrdered(i) = itemsType(k)
     ItemsOrdered(i) = items(k)
     ItemsOrdered2(i) = items2(k)
     demand(k) = demand(k) - 1

     AnItemHasBeenSelected = True
     SelectedIn2 = True
     Exit For

   ElseIf (r > CDF(k) And r <= CDF(k + 1) And demand(k) = 0 And AnItemHasBeenSelected = False) Then
    For t = k + 1 To UBound(items) - 1
        If (demand(t) > 0) Then
          ItemsTypeOrdered(i) = itemsType(t)
          ItemsOrdered(i) = items(t)
          ItemsOrdered2(i) = items2(t)
          demand(t) = demand(t) - 1
          AnItemHasBeenSelected = True
          SelectedIn2 = True
          Exit For
        End If

    Next t
    Exit For
   End If

Next k
End If

 If (Not SelectedIn1 And Not SelectedIn2) Then

 r = Rnd
```



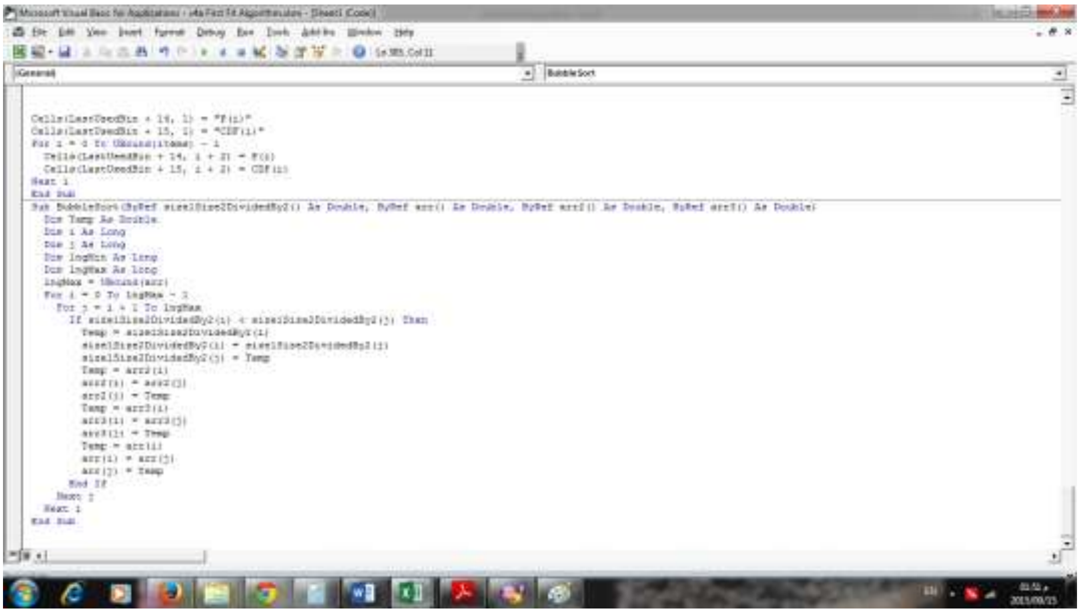
```
 If (Not SelectedIn1 And Not SelectedIn2) Then

 r = Rnd
 i = i - 1
End If

Next i

ReDim Bins(SumDemands) As clsBin

For i = 0 To SumDemands
   Set Bins(i) = New clsBin
   Bins(i).updateLastUsedPlace = 0
   Bins(i).updateSize = 0
   Bins(i).updateSize2 = 0

Next i

Dim LastUsedBin As Double


LastUsedBin = 0
For i = 0 To SumDemands - 1

  foundAplaceForTheItem = False


  For j = 0 To LastUsedBin
   If (Bins(j).sizeOfBin + ItemsOrdered(i) <= MaximumSizeOfABin And Bins(j).size2OfBin + ItemsOrdered2(i) <= MaximumSize2OfABin) Then

      Bins(j).updateSize = Bins(j).sizeOfBin + ItemsOrdered(i)
      Bins(j).updateSize2 = Bins(j).size2OfBin + ItemsOrdered2(i)
      Bins(j).updateLastUsedPlace = Bins(j).LastUsedPlaceOfBin + 1
      Cells(j + 8, Bins(j).LastUsedPlaceOfBin + 1) = ItemsTypeOrdered(i)
```



```
      Bins(j).updateSize = Bins(j).sizeOfBin + ItemsOrdered(i)
      Bins(j).updateSize2 = Bins(j).size2OfBin + ItemsOrdered2(i)
      Bins(j).updateLastUsedPlace = Bins(j).LastUsedPlaceOfBin + 1
      Cells(j + 8, Bins(j).LastUsedPlaceOfBin + 1) = ItemsTypeOrdered(i)
      foundAplaceForTheItem = True
      Exit For
    End If

  Next j
  If (Not foundAplaceForTheItem) Then
    LastUsedBin = LastUsedBin + 1
    Cells(LastUsedBin + 8, 1) = "Bin " & CStr(LastUsedBin + 1)


    Bins(LastUsedBin).updateSize = Bins(LastUsedBin).sizeOfBin + ItemsOrdered(i)
    Bins(LastUsedBin).updateSize2 = Bins(LastUsedBin).size2OfBin + ItemsOrdered2(i)
    Bins(LastUsedBin).updateLastUsedPlace = Bins(LastUsedBin).LastUsedPlaceOfBin + 1

    Cells(LastUsedBin + 8, Bins(LastUsedBin).LastUsedPlaceOfBin + 1) = ItemsTypeOrdered(i)
    foundAplaceForTheItem = True


  End If

Next i

Dim dTotal As Double
Dim dTotal2 As Double
Dim dAverage As Double
Dim dAverage2 As Double

dTotal = 0
dTotal2 = 0
```

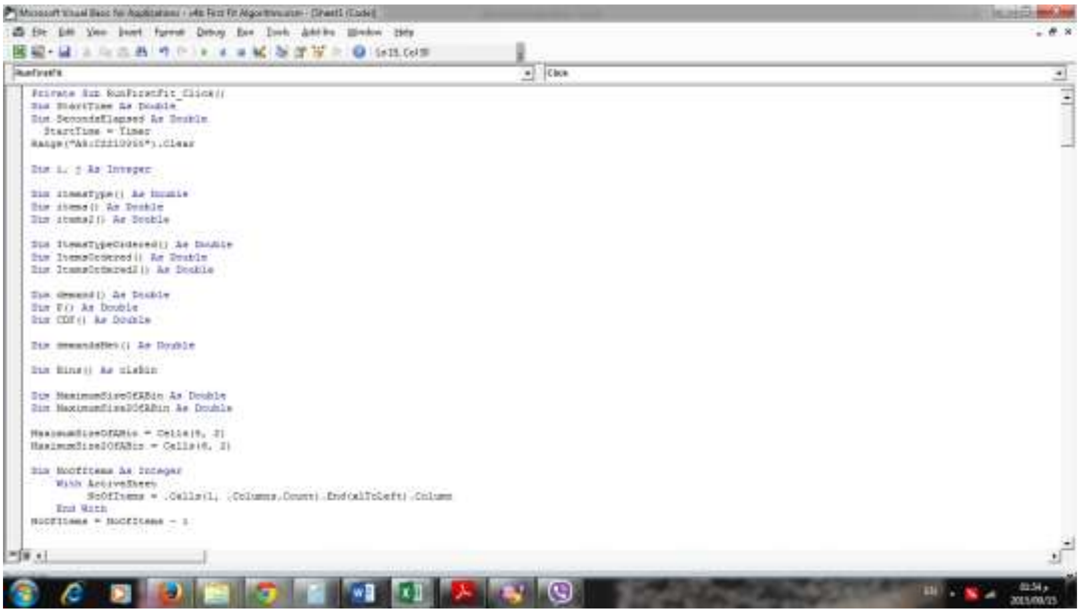


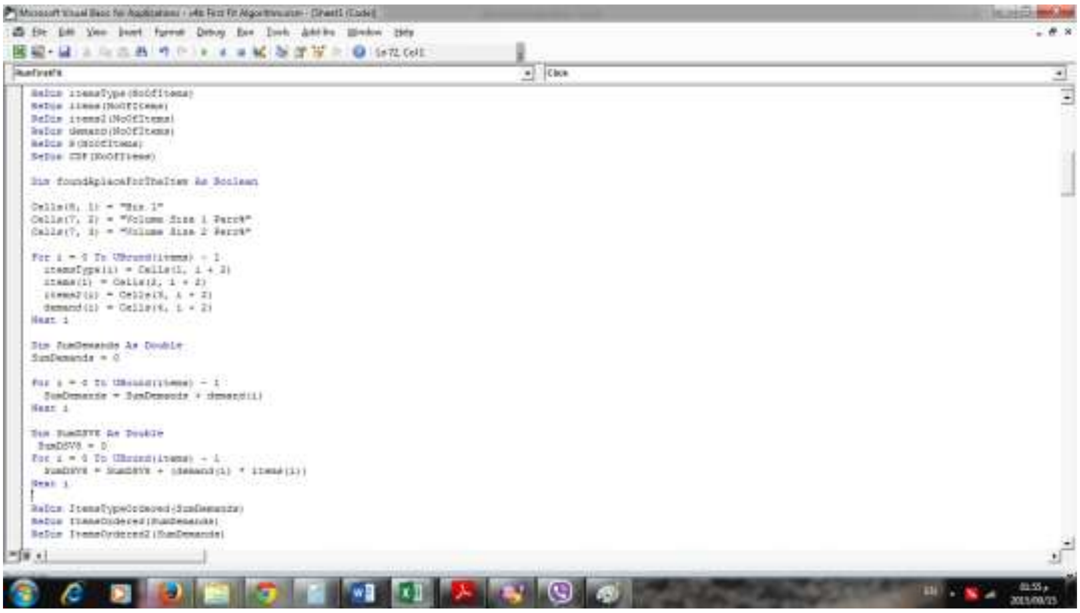
```
For j = 0 To LastUsedBin

        Cells(j + 8, 2) = Round(Bins(j).sizeOfBin / MaximumSizeOfABin * 100, 3)
        Cells(j + 8, 3) = Round(Bins(j).size2OfBin / MaximumSize2OfABin * 100, 3)

         dTotal = dTotal + (Bins(j).sizeOfBin / MaximumSizeOfABin * 100)
         dTotal2 = dTotal2 + (Bins(j).size2OfBin / MaximumSize2OfABin * 100)
Next j

dAverage = dTotal / (LastUsedBin + 1)
dAverage2 = dTotal2 / (LastUsedBin + 1)

Cells(LastUsedBin + 9, 1) = "Avg Volume Size"
Cells(LastUsedBin + 9, 2) = "Avg Volume Size2"


Cells(LastUsedBin + 10, 1) = dAverage
Cells(LastUsedBin + 10, 2) = dAverage2


Cells(LastUsedBin + 11, 1) = "Run Time (in Seconds)"
SecondsElapsed = Round(Timer - StartTime, 2)
Cells(LastUsedBin + 11, 2) = SecondsElapsed

Cells(LastUsedBin + 12, 1) = "Size1 after random ordering"
Cells(LastUsedBin + 13, 1) = "Size2 after random ordering"


For i = 0 To NumDemands - 1

  Cells(LastUsedBin + 12, i + 2) = ItemsOrdered(i)
  Cells(LastUsedBin + 13, i + 2) = ItemsOrdered2(i)

Next i
```

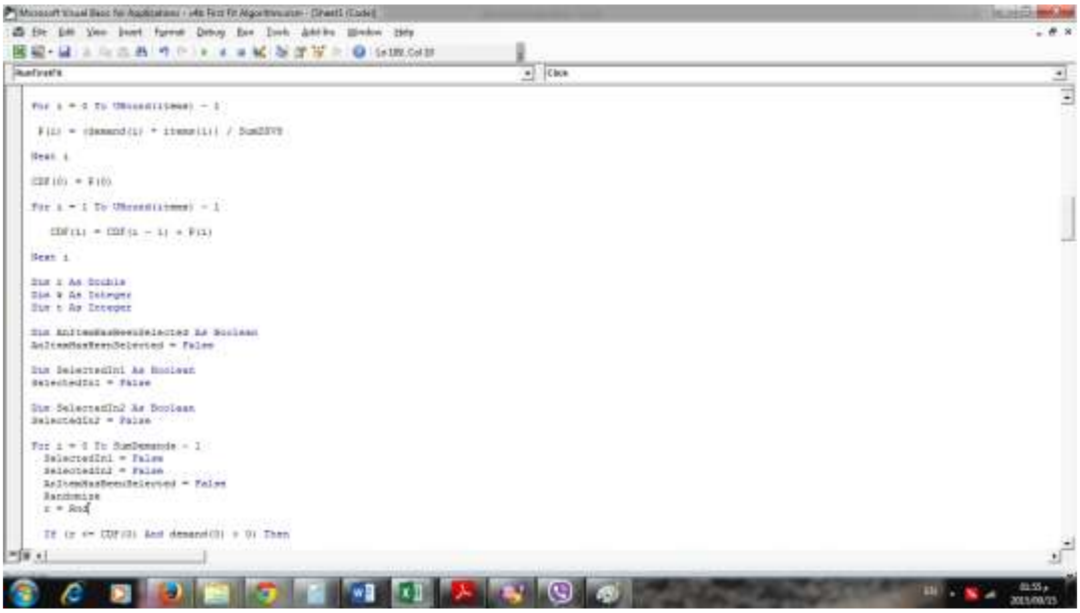


```
Cells(LastUsedBin + 14, 1) = "F(i)"
Cells(LastUsedBin + 15, 1) = "CDF(i)"
For i = 0 To UBound(items) - 1
  Cells(LastUsedBin + 14, i + 2) = F(i)
  Cells(LastUsedBin + 15, i + 2) = CDF(i)
Next i
End Sub
Sub BubbleSort(ByRef size1Size2DividedBy2() As Double, ByRef arr() As Double, ByRef arr2() As Double, ByRef arr3() As Double)
  Dim Temp As Double
  Dim i As Long
  Dim j As Long
  Dim lngMin As Long
  Dim lngMax As Long
  lngMax = UBound(arr)
  For i = 0 To lngMax - 1
    For j = i + 1 To lngMax
      If size1Size2DividedBy2(i) < size1Size2DividedBy2(j) Then
        Temp = size1Size2DividedBy2(i)
        size1Size2DividedBy2(i) = size1Size2DividedBy2(j)
        size1Size2DividedBy2(j) = Temp
        Temp = arr2(i)
        arr2(i) = arr2(j)
        arr2(j) = Temp
        Temp = arr3(i)
        arr3(i) = arr3(j)
        arr3(j) = Temp
        Temp = arr(i)
        arr(i) = arr(j)
        arr(j) = Temp
      End If
    Next j
  Next i
End Sub
```

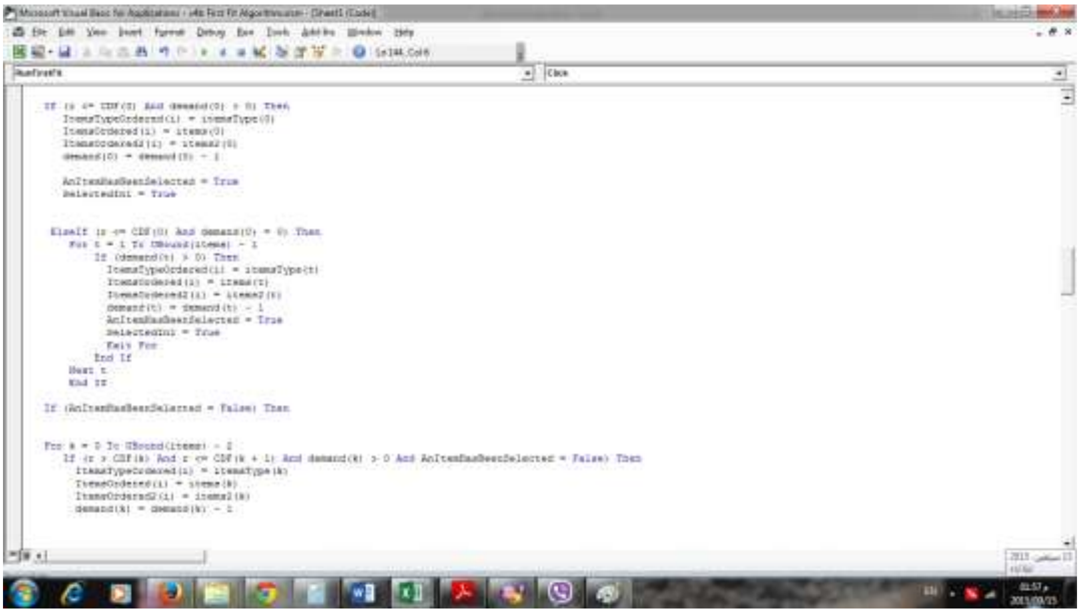
This is the configuration code of the algorithm 6



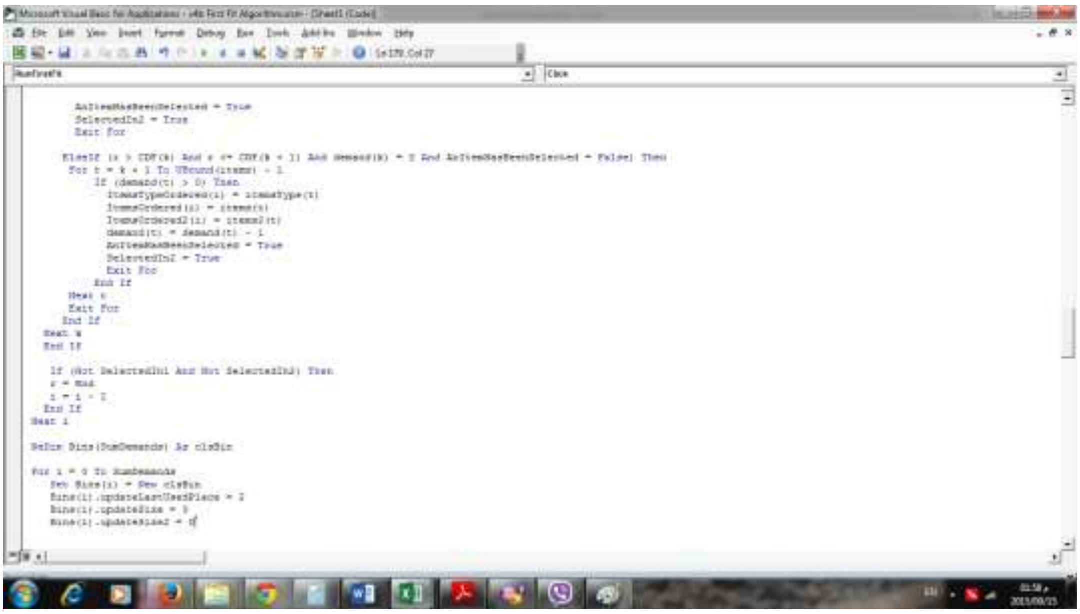




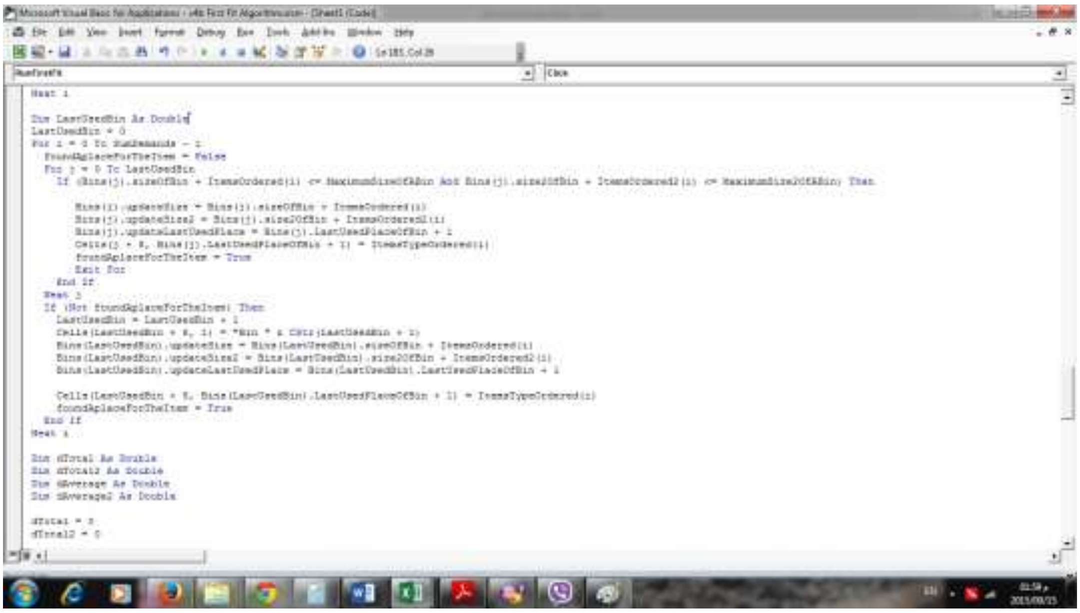
```
For i = 0 To UBound(items) - 1

 F(i) = (demand(i) * items(i)) / SumDSVS

Next i

CDF(0) = F(0)

For i = 1 To UBound(items) - 1

   CDF(i) = CDF(i - 1) + F(i)

Next i

Dim r As Double
Dim k As Integer
Dim t As Integer

Dim AnItemHasBeenSelected As Boolean
AnItemHasBeenSelected = False

Dim SelectedIn1 As Boolean
SelectedIn1 = False

Dim SelectedIn2 As Boolean
SelectedIn2 = False

For i = 0 To SumDemands - 1
  SelectedIn1 = False
  SelectedIn2 = False
  AnItemHasBeenSelected = False
  Randomize
  r = Rnd

  If (r <= CDF(0) And demand(0) > 0) Then
```



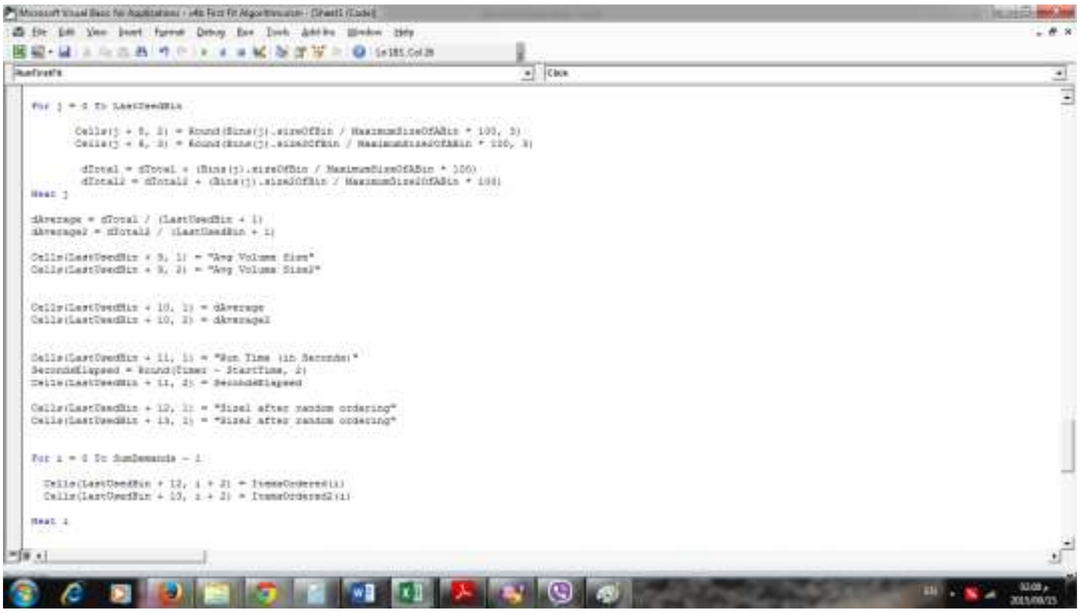
```
  If (r <= CDF(0) And demand(0) > 0) Then
     ItemsTypeOrdered(i) = itemsType(0)
     ItemsOrdered(i) = items(0)
     ItemsOrdered2(i) = items2(0)
     demand(0) = demand(0) - 1

     AnItemHasBeenSelected = True
     SelectedIn1 = True


   ElseIf (r <= CDF(0) And demand(0) = 0) Then
      For t = 1 To UBound(items) - 1
         If (demand(t) > 0) Then
           ItemsTypeOrdered(i) = itemsType(t)
           ItemsOrdered(i) = items(t)
           ItemsOrdered2(i) = items2(t)
           demand(t) = demand(t) - 1
           AnItemHasBeenSelected = True
           SelectedIn1 = True
           Exit For
         End If
      Next t
      End If

  If (AnItemHasBeenSelected = False) Then


  For k = 0 To UBound(items) - 2
     If (r > CDF(k) And r <= CDF(k + 1) And demand(k) > 0 And AnItemHasBeenSelected = False) Then
       ItemsTypeOrdered(i) = itemsType(k)
       ItemsOrdered(i) = items(k)
       ItemsOrdered2(i) = items2(k)
       demand(k) = demand(k) - 1
```



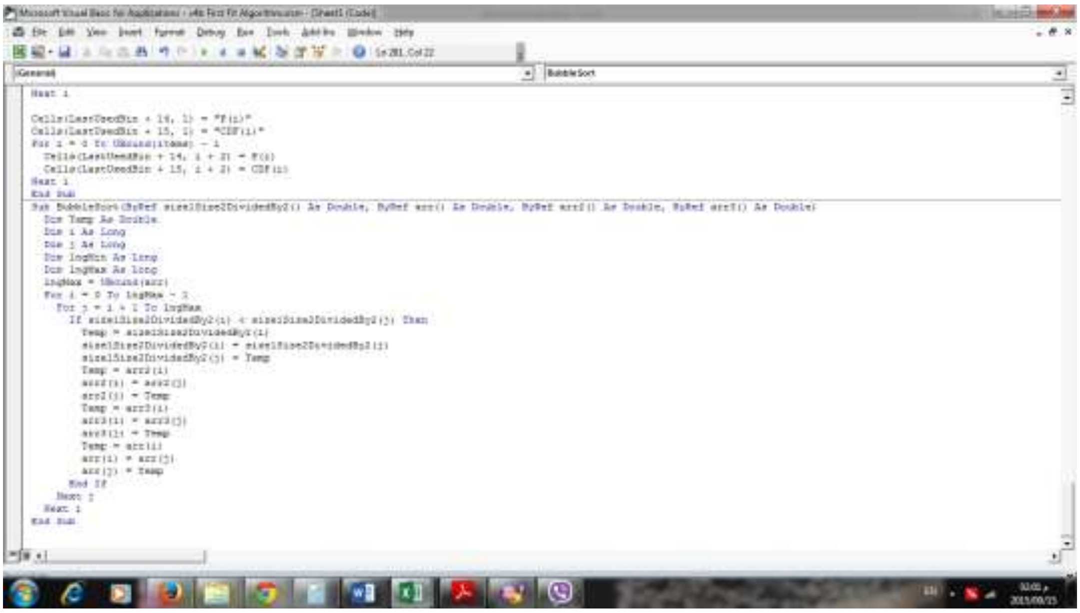
```
        AnItemHasBeenSelected = True
        SelectedIn2 = True
        Exit For

     ElseIf (r > CDF(k) And r <= CDF(k + 1) And demand(k) = 0 And AnItemHasBeenSelected = False) Then
      For t = k + 1 To UBound(items) - 1
         If (demand(t) > 0) Then
           ItemsTypeOrdered(i) = itemsType(t)
           ItemsOrdered(i) = items(t)
           ItemsOrdered2(i) = items2(t)
           demand(t) = demand(t) - 1
           AnItemHasBeenSelected = True
           SelectedIn2 = True
           Exit For
         End If
      Next t
      Exit For
     End If
  Next k
  End If

   If (Not SelectedIn1 And Not SelectedIn2) Then
   r = Rnd
   i = i - 1
  End If
Next i

ReDim Bins(SumDemands) As clsBin

For i = 0 To SumDemands
   Set Bins(i) = New clsBin
   Bins(i).updateLastUsedPlace = 1
   Bins(i).updateSize = 0
   Bins(i).updateSize2 = 0
```



```
Next i

Dim LastUsedBin As Double
LastUsedBin = 0
For i = 0 To NumDemands - 1
  foundAplaceForTheItem = False
  For j = 0 To LastUsedBin
    If (Bins(j).sizeOfBin + ItemsOrdered(i) <= MaximumSizeOfABin And Bins(j).size2OfBin + ItemsOrdered2(i) <= MaximumSize2OfABin) Then

      Bins(j).updateSize = Bins(j).sizeOfBin + ItemsOrdered(i)
      Bins(j).updateSize2 = Bins(j).size2OfBin + ItemsOrdered2(i)
      Bins(j).updateLastUsedPlace = Bins(j).LastUsedPlaceOfBin + 1
      Cells(j + 8, Bins(j).LastUsedPlaceOfBin + 1) = ItemsTypeOrdered(i)
      foundAplaceForTheItem = True
      Exit For
    End If
  Next j
  If (Not foundAplaceForTheItem) Then
    LastUsedBin = LastUsedBin + 1
    Cells(LastUsedBin + 8, 1) = "Bin " & CStr(LastUsedBin + 1)
    Bins(LastUsedBin).updateSize = Bins(LastUsedBin).sizeOfBin + ItemsOrdered(i)
    Bins(LastUsedBin).updateSize2 = Bins(LastUsedBin).size2OfBin + ItemsOrdered2(i)
    Bins(LastUsedBin).updateLastUsedPlace = Bins(LastUsedBin).LastUsedPlaceOfBin + 1

    Cells(LastUsedBin + 8, Bins(LastUsedBin).LastUsedPlaceOfBin + 1) = ItemsTypeOrdered(i)
    foundAplaceForTheItem = True
  End If
Next i

Dim dTotal As Double
Dim dTotal2 As Double
Dim dAverage As Double
Dim dAverage2 As Double

dTotal = 0
dTotal2 = 0
```

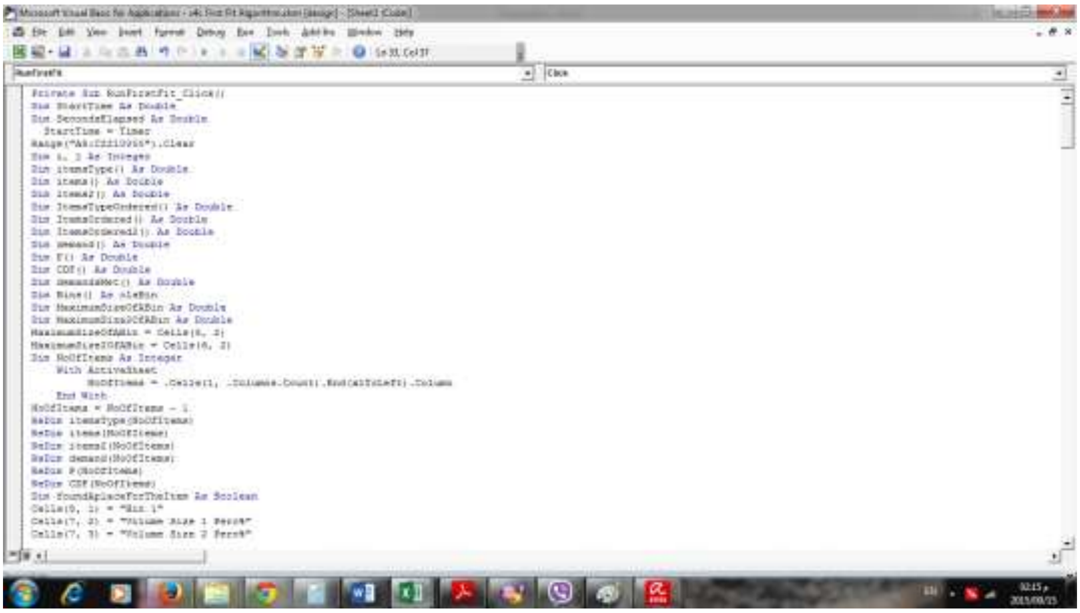


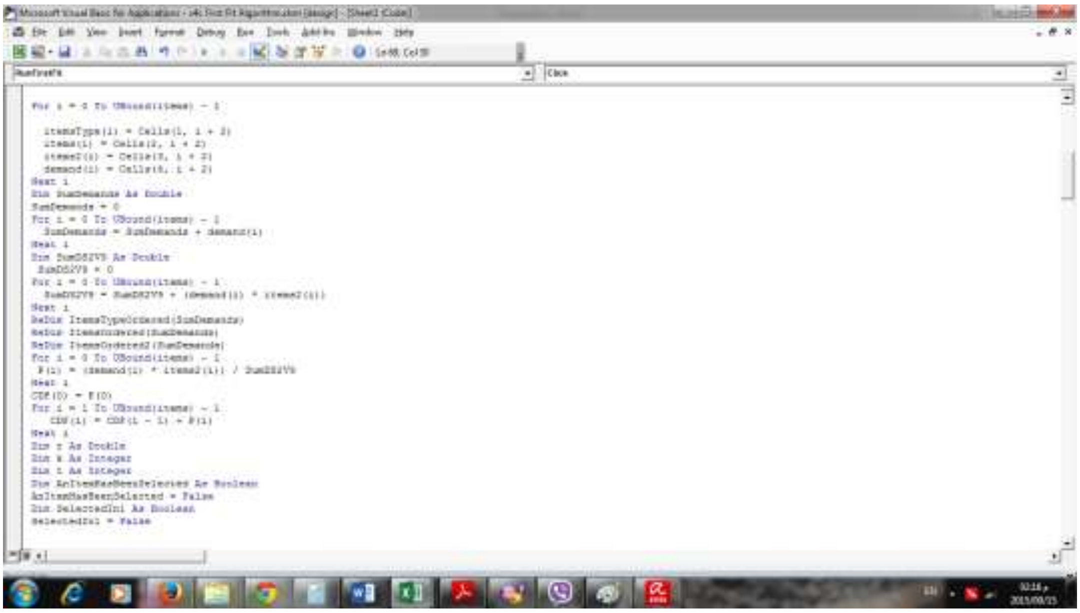
```
For j = 0 To LastUsedBin

      Cells(j + 8, 2) = Round(Bins(j).sizeOfBin / MaximumSizeOfABin * 100, 3)
      Cells(j + 8, 3) = Round(Bins(j).size2OfBin / MaximumSize2OfABin * 100, 3)

       dTotal = dTotal + (Bins(j).sizeOfBin / MaximumSizeOfABin * 100)
       dTotal2 = dTotal2 + (Bins(j).size2OfBin / MaximumSize2OfABin * 100)
Next j

dAverage = dTotal / (LastUsedBin + 1)
dAverage2 = dTotal2 / (LastUsedBin + 1)

Cells(LastUsedBin + 9, 1) = "Avg Volume Size"
Cells(LastUsedBin + 9, 2) = "Avg Volume Size2"


Cells(LastUsedBin + 10, 1) = dAverage
Cells(LastUsedBin + 10, 2) = dAverage2


Cells(LastUsedBin + 11, 1) = "Run Time (in Seconds)"
SecondsElapsed = Round(Timer - StartTime, 2)
Cells(LastUsedBin + 11, 2) = SecondsElapsed

Cells(LastUsedBin + 12, 1) = "Size1 after random ordering"
Cells(LastUsedBin + 13, 1) = "Size2 after random ordering"


For i = 0 To NumDemands - 1

  Cells(LastUsedBin + 12, i + 2) = ItemsOrdered(i)
  Cells(LastUsedBin + 13, i + 2) = ItemsOrdered2(i)

Next i
```

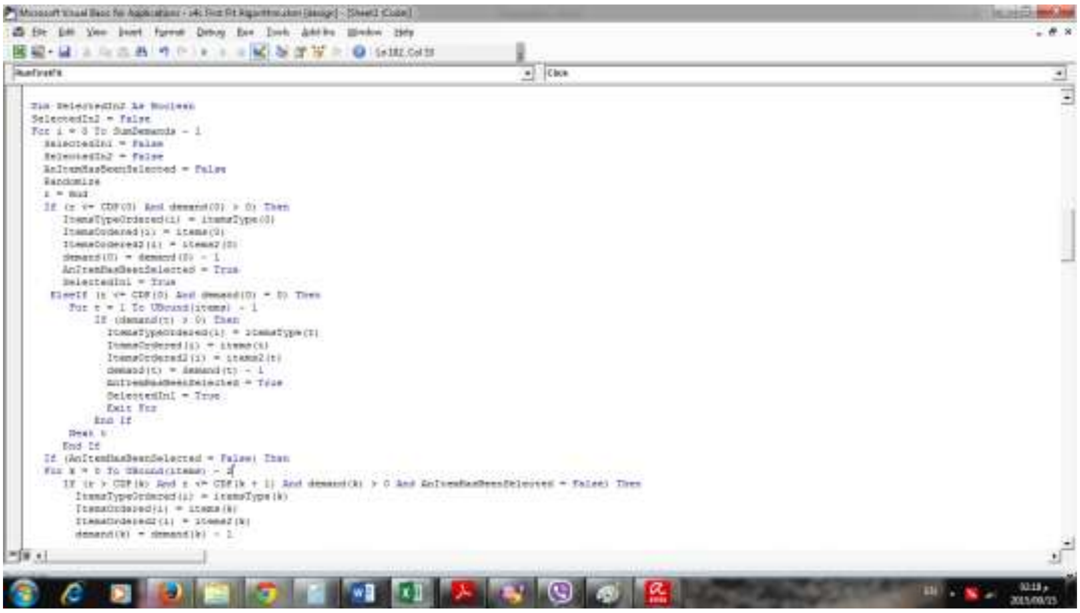


```
Next i

Cells(LastUsedBin + 14, 1) = "F(i)"
Cells(LastUsedBin + 15, 1) = "CDF(i)"
For i = 0 To UBound(items) - 1
  Cells(LastUsedBin + 14, i + 2) = F(i)
  Cells(LastUsedBin + 15, i + 2) = CDF(i)
Next i
End Sub
Sub BubbleSort(ByRef size1Size2DividedBy2() As Double, ByRef arr() As Double, ByRef arr2() As Double, ByRef arr3() As Double)
  Dim Temp As Double
  Dim i As Long
  Dim j As Long
  Dim lngMin As Long
  Dim lngMax As Long
  lngMax = UBound(arr)
  For i = 0 To lngMax - 1
    For j = i + 1 To lngMax
      If size1Size2DividedBy2(i) < size1Size2DividedBy2(j) Then
        Temp = size1Size2DividedBy2(i)
        size1Size2DividedBy2(i) = size1Size2DividedBy2(j)
        size1Size2DividedBy2(j) = Temp
        Temp = arr2(i)
        arr2(i) = arr2(j)
        arr2(j) = Temp
        Temp = arr3(i)
        arr3(i) = arr3(j)
        arr3(j) = Temp
        Temp = arr(i)
        arr(i) = arr(j)
        arr(j) = Temp
      End If
    Next j
  Next i
End Sub
```

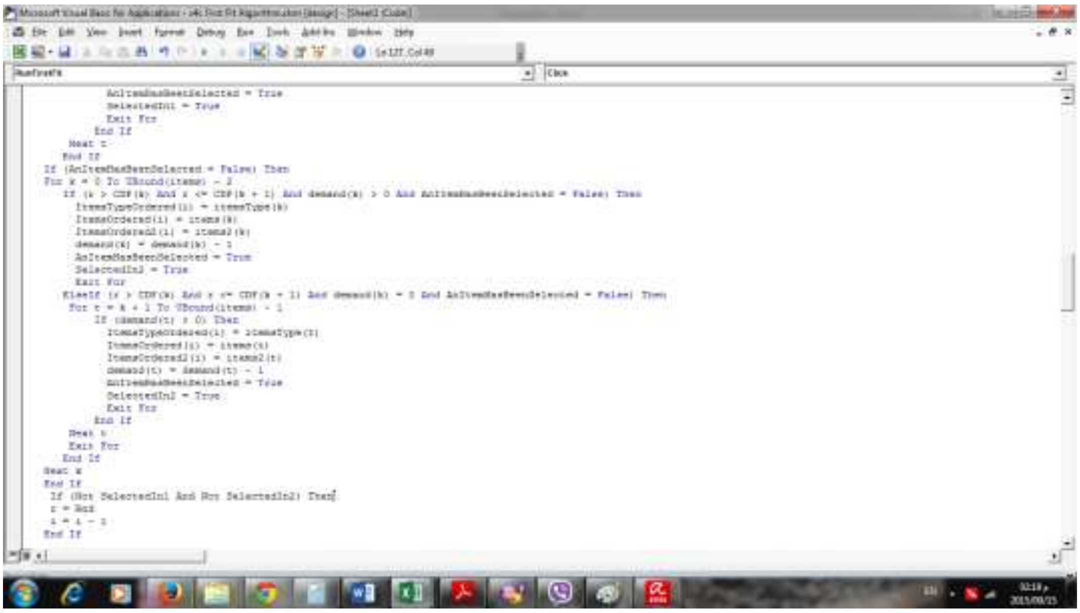
This is the configuration code of the algorithm 7



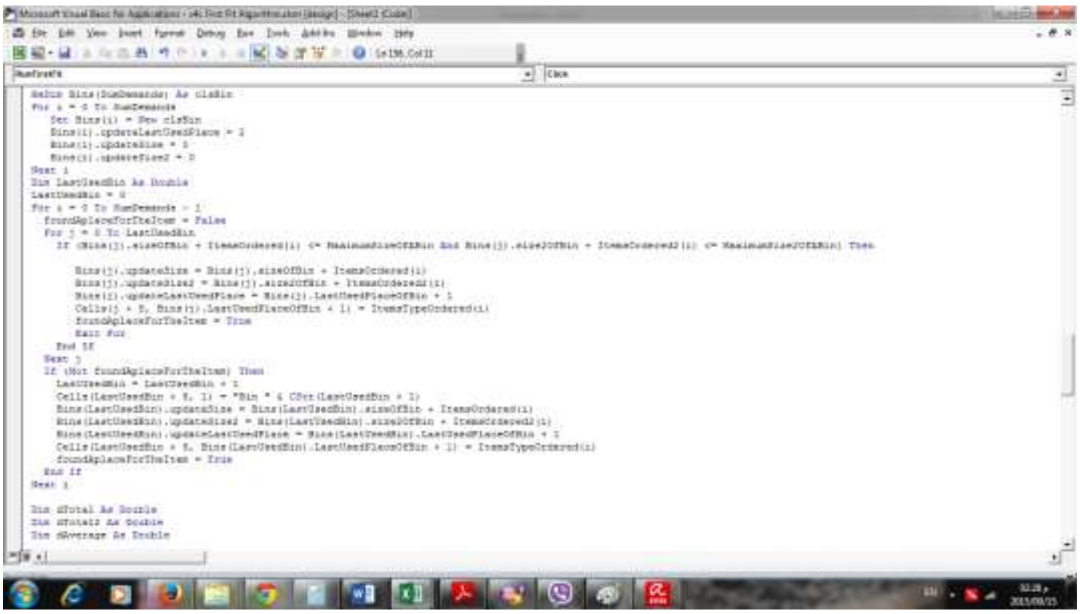




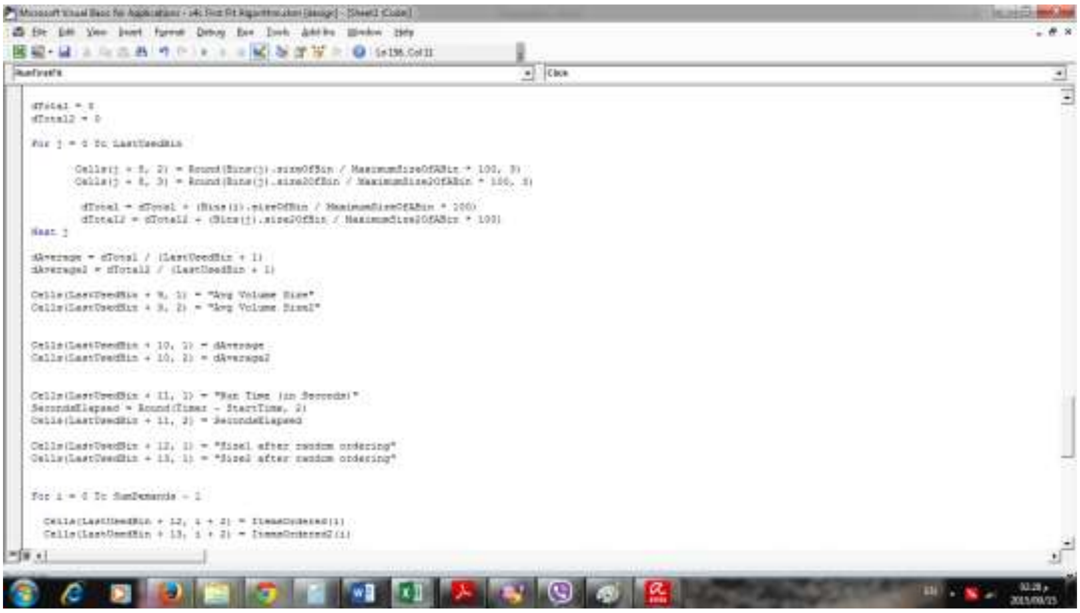
```
Dim SelectedIn2 As Boolean
SelectedIn2 = False
For i = 0 To SumDemands - 1
  SelectedIn1 = False
  SelectedIn2 = False
  AnItemHasBeenSelected = False
  Randomize
  r = Rnd
  If (r <= CDF(0) And demand(0) > 0) Then
    ItemsTypeOrdered(i) = itemsType(0)
    ItemsOrdered(i) = items(0)
    ItemsOrdered2(i) = items2(0)
    demand(0) = demand(0) - 1
    AnItemHasBeenSelected = True
    SelectedIn1 = True
   ElseIf (r <= CDF(0) And demand(0) = 0) Then
     For t = 1 To UBound(items) - 1
         If (demand(t) > 0) Then
           ItemsTypeOrdered(i) = itemsType(t)
           ItemsOrdered(i) = items(t)
           ItemsOrdered2(i) = items2(t)
           demand(t) = demand(t) - 1
           AnItemHasBeenSelected = True
           SelectedIn1 = True
           Exit For
         End If
     Next t
    End If
  If (AnItemHasBeenSelected = False) Then
  For k = 0 To UBound(items) - 2
     If (r > CDF(k) And r <= CDF(k + 1) And demand(k) > 0 And AnItemHasBeenSelected = False) Then
      ItemsTypeOrdered(i) = itemsType(k)
      ItemsOrdered(i) = items(k)
      ItemsOrdered2(i) = items2(k)
      demand(k) = demand(k) - 1
```

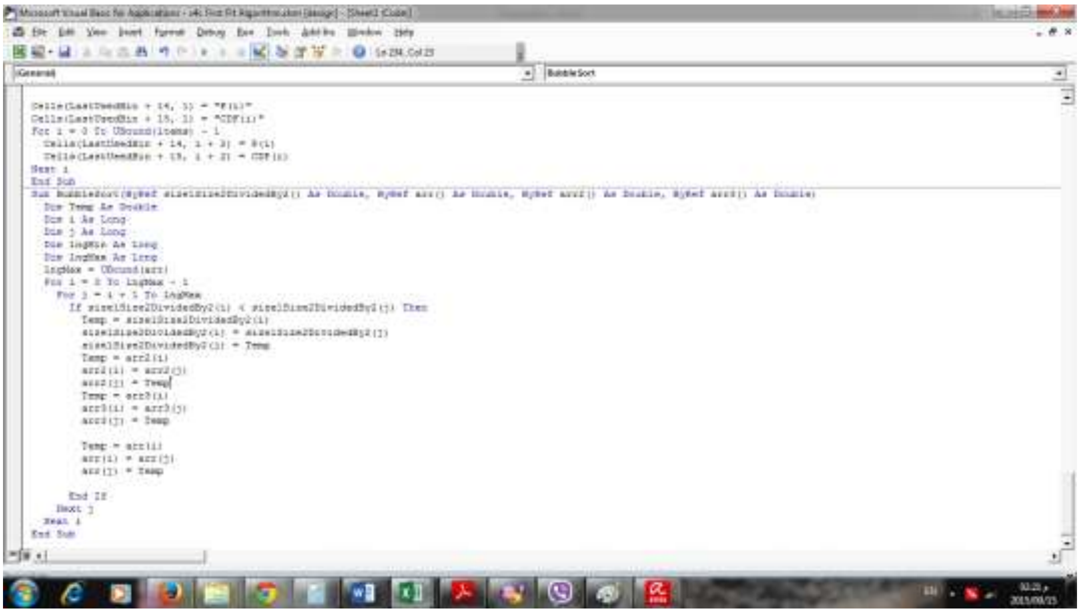


```
           AnItemHasBeenSelected = True
           SelectedIn1 = True
           Exit For
         End If
     Next t
    End If
  If (AnItemHasBeenSelected = False) Then
  For k = 0 To UBound(items) - 2
     If (r > CDF(k) And r <= CDF(k + 1) And demand(k) > 0 And AnItemHasBeenSelected = False) Then
      ItemsTypeOrdered(i) = itemsType(k)
      ItemsOrdered(i) = items(k)
      ItemsOrdered2(i) = items2(k)
      demand(k) = demand(k) - 1
      AnItemHasBeenSelected = True
      SelectedIn2 = True
      Exit For
    ElseIf (r > CDF(k) And r <= CDF(k + 1) And demand(k) = 0 And AnItemHasBeenSelected = False) Then
     For t = k + 1 To UBound(items) - 1
         If (demand(t) > 0) Then
           ItemsTypeOrdered(i) = itemsType(t)
           ItemsOrdered(i) = items(t)
           ItemsOrdered2(i) = items2(t)
           demand(t) = demand(t) - 1
           AnItemHasBeenSelected = True
           SelectedIn2 = True
           Exit For
         End If
     Next t
     Exit For
    End If
  Next k
  End If
   If (Not SelectedIn1 And Not SelectedIn2) Then
   r = Rnd
   i = i - 1
  End If
```

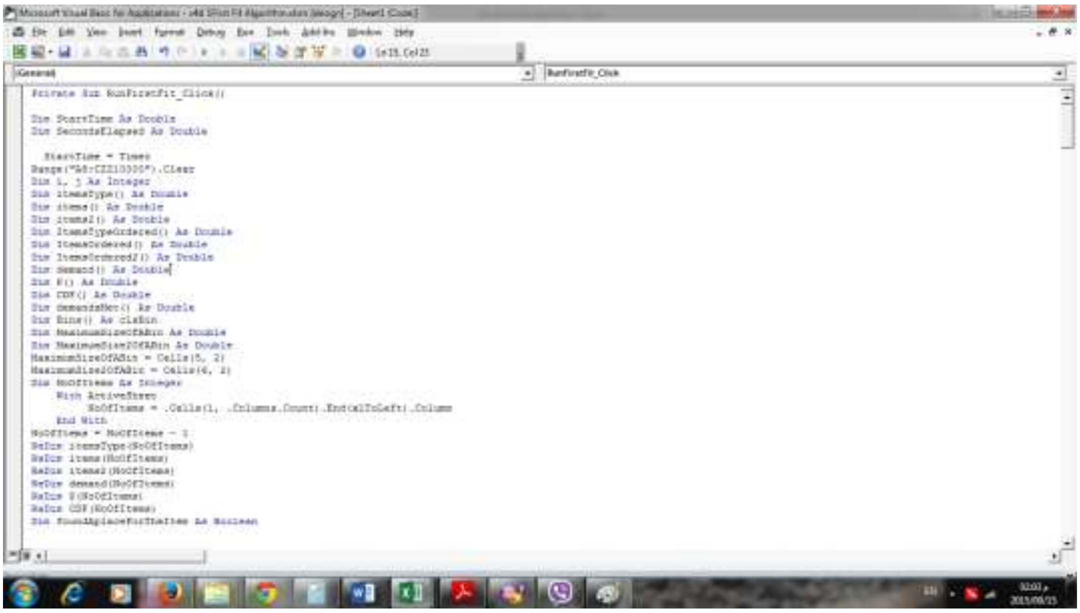


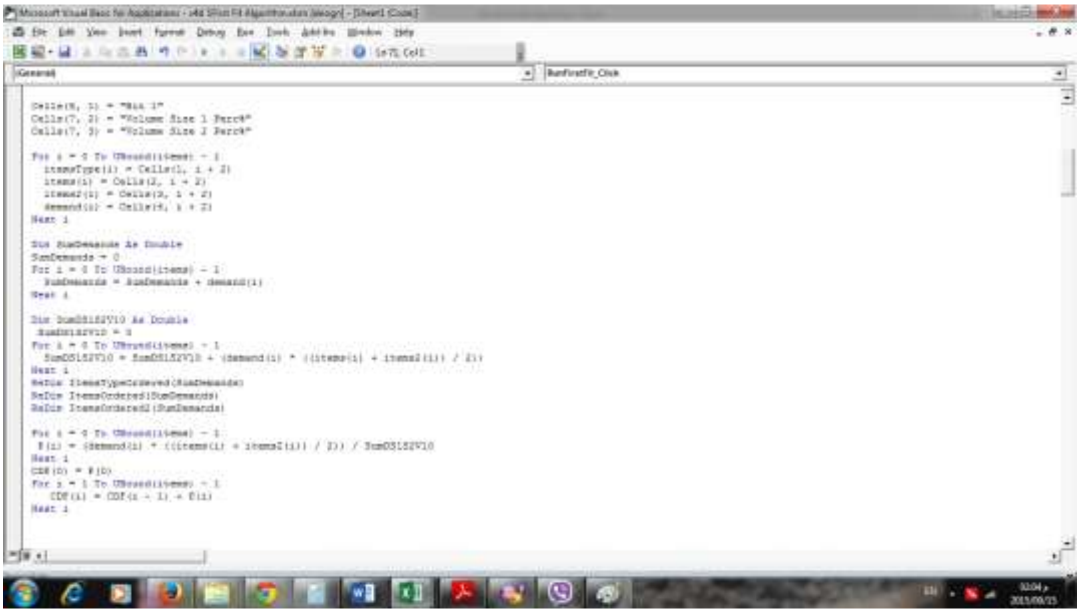
```
ReDim Bins(SumDemands) As clsBin
For i = 0 To SumDemands
   Set Bins(i) = New clsBin
   Bins(i).updateLastUsedPlace = 2
   Bins(i).updateSize = 0
   Bins(i).updateSize2 = 0
Next i
Dim LastUsedBin As Double
LastUsedBin = 0
For i = 0 To SumDemands - 1
  foundAplaceForTheItem = False
  For j = 0 To LastUsedBin
    If (Bins(j).sizeOfBin + ItemsOrdered(i) <= MaximumSizeOfEBin And Bins(j).size2OfBin + ItemsOrdered2(i) <= MaximumSize2OfEBin) Then

      Bins(j).updateSize = Bins(j).sizeOfBin + ItemsOrdered(i)
      Bins(j).updateSize2 = Bins(j).size2OfBin + ItemsOrdered2(i)
      Bins(j).updateLastUsedPlace = Bins(j).LastUsedPlaceOfBin + 1
      Cells(j + 5, Bins(j).LastUsedPlaceOfBin + 1) = ItemsTypeOrdered(i)
      foundAplaceForTheItem = True
      Exit For
    End If
  Next j
  If (Not foundAplaceForTheItem) Then
    LastUsedBin = LastUsedBin + 1
    Cells(LastUsedBin + 5, 1) = "Bin " & CStr(LastUsedBin + 1)
    Bins(LastUsedBin).updateSize = Bins(LastUsedBin).sizeOfBin + ItemsOrdered(i)
    Bins(LastUsedBin).updateSize2 = Bins(LastUsedBin).size2OfBin + ItemsOrdered2(i)
    Bins(LastUsedBin).updateLastUsedPlace = Bins(LastUsedBin).LastUsedPlaceOfBin + 1
    Cells(LastUsedBin + 5, Bins(LastUsedBin).LastUsedPlaceOfBin + 1) = ItemsTypeOrdered(i)
    foundAplaceForTheItem = True
  End If
Next i

Dim dTotal As Double
Dim dTotal2 As Double
Dim dAverage As Double
```

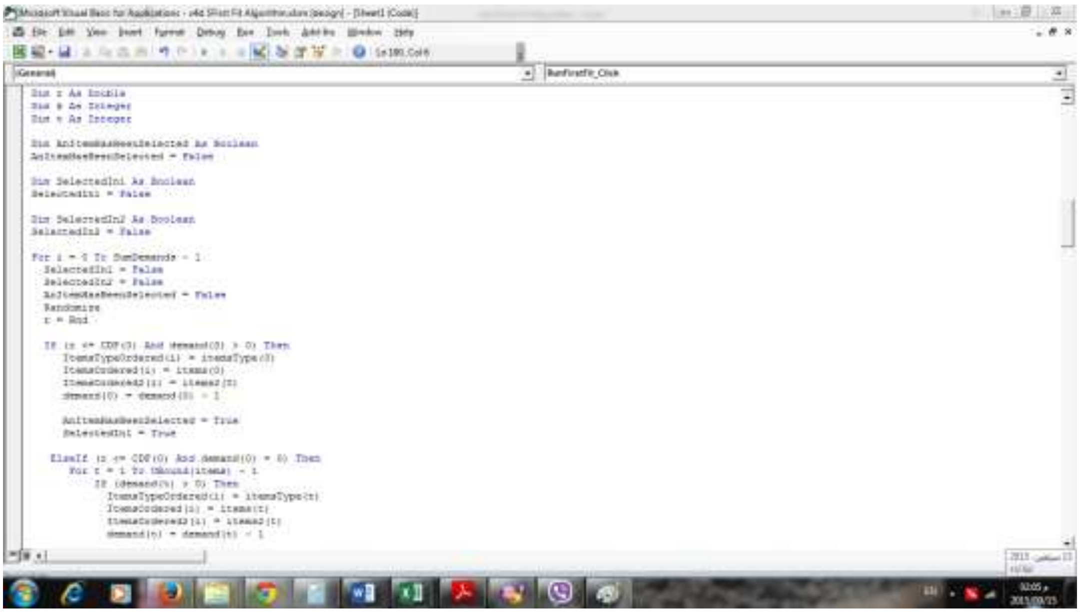

dTotal = 0
dTotal2 = 0
For j = 0 To LastUsedBin
Cells(j + 8, 2) = Round(Bins(j).sizeOfBin / MaximumSizeOfABin * 100, 3)
Cells(j + 8, 3) = Round(Bins(j).size2OfBin / MaximumSize2OfABin * 100, 3)
dTotal = dTotal + (Bins(j).sizeOfBin / MaximumSizeOfABin * 100)
dTotal2 = dTotal2 + (Bins(j).size2OfBin / MaximumSize2OfABin * 100)
Next j
dAverage = dTotal / (LastUsedBin + 1)
dAverage2 = dTotal2 / (LastUsedBin + 1)
Cells(LastUsedBin + 9, 1) = "Avg Volume Size"
Cells(LastUsedBin + 9, 2) = "Avg Volume Size2"
Cells(LastUsedBin + 10, 1) = dAverage
Cells(LastUsedBin + 10, 2) = dAverage2
Cells(LastUsedBin + 11, 1) = "Run Time (in Seconds)"
SecondsElapsed = Round(Timer - StartTime, 2)
Cells(LastUsedBin + 11, 2) = SecondsElapsed
Cells(LastUsedBin + 12, 1) = "Size1 after random ordering"
Cells(LastUsedBin + 13, 1) = "Size2 after random ordering"
For i = 0 To SumDemands - 1
Cells(LastUsedBin + 12, i + 2) = ItemsOrdered(i)
Cells(LastUsedBin + 13, i + 2) = ItemsOrdered2(i)


Cells(LastUsedBin + 14, 1) = "F(i)"
Cells(LastUsedBin + 15, 1) = "CDF(i)"
For i = 0 To UBound(items) - 1
Cells(LastUsedBin + 14, i + 2) = F(i)
Cells(LastUsedBin + 15, i + 2) = CDF(i)
Next i
End Sub
Sub BubbleSort(ByRef size1Size2DividedBy2() As Double, ByRef arr() As Double, ByRef arr2() As Double, ByRef arr3() As Double)
Dim Temp As Double
Dim i As Long
Dim j As Long
Dim lngMin As Long
Dim lngMax As Long
lngMax = UBound(arr)
For i = 0 To lngMax - 1
For j = i + 1 To lngMax
If size1Size2DividedBy2(i) < size1Size2DividedBy2(j) Then
Temp = size1Size2DividedBy2(i)
size1Size2DividedBy2(i) = size1Size2DividedBy2(j)
size1Size2DividedBy2(j) = Temp
Temp = arr2(i)
arr2(i) = arr2(j)
arr2(j) = Temp
Temp = arr3(i)
arr3(i) = arr3(j)
arr3(j) = Temp
Temp = arr(i)
arr(i) = arr(j)
arr(j) = Temp
End If
Next j
Next i
End Sub

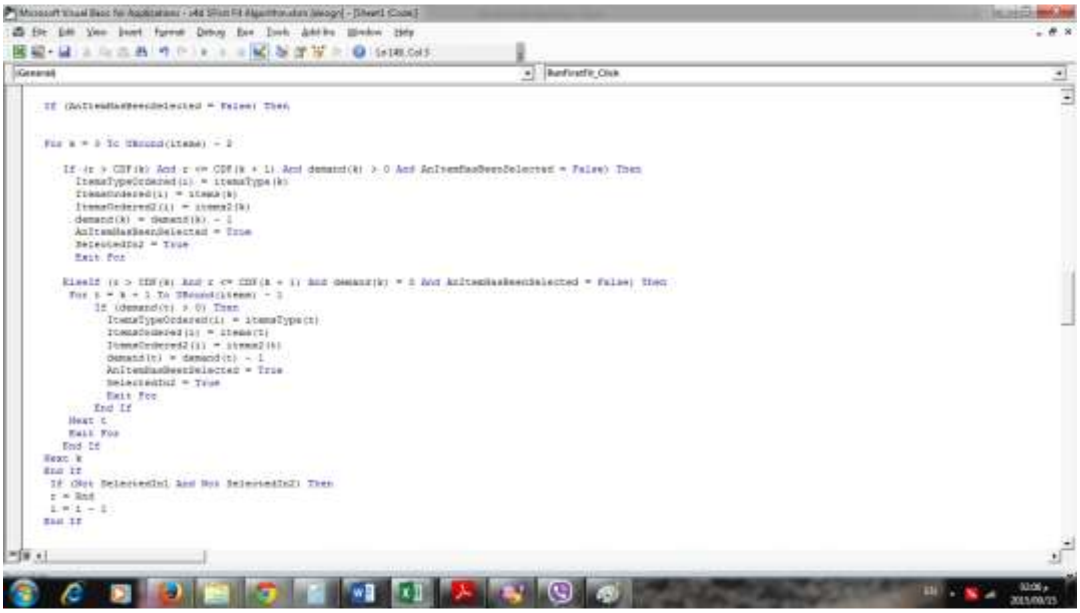
This is the configuration code of the algorithm 8



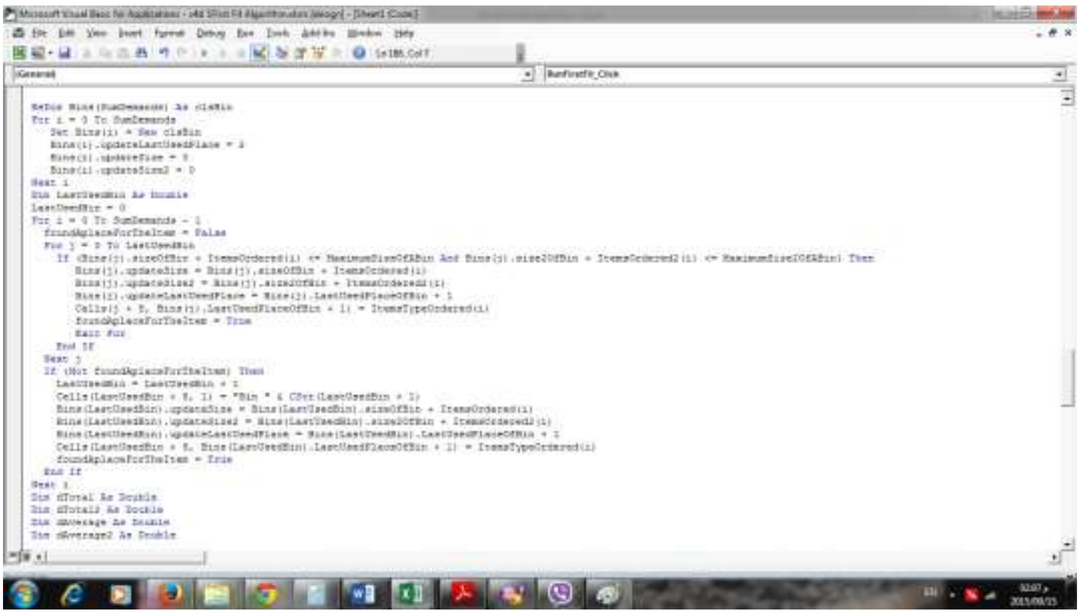




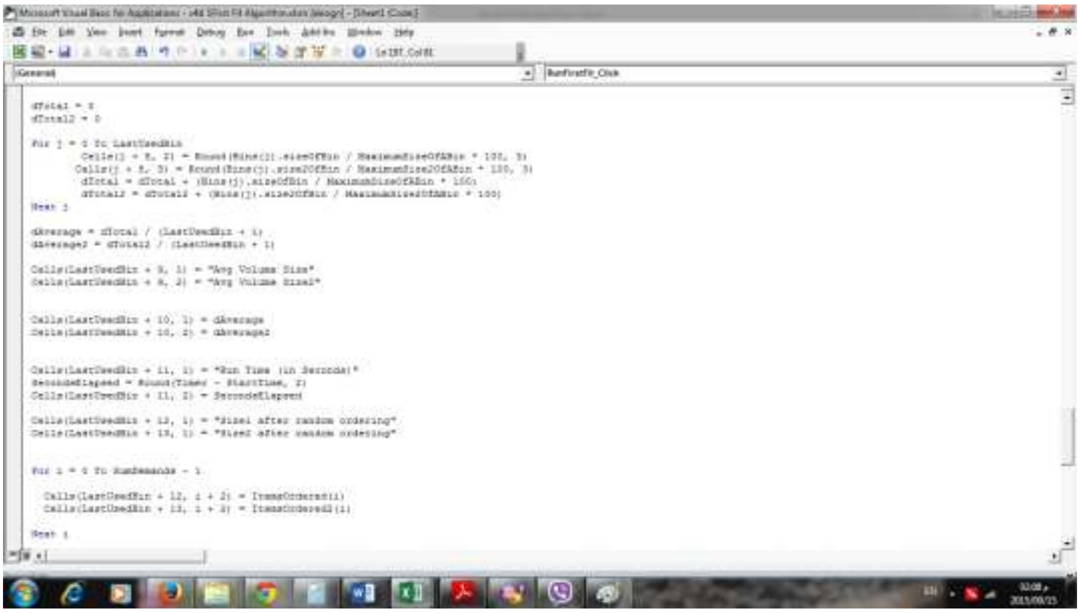
```
Dim r As Double
Dim k As Integer
Dim t As Integer

Dim AnItemHasBeenSelected As Boolean
AnItemHasBeenSelected = False

Dim SelectedIn1 As Boolean
SelectedIn1 = False

Dim SelectedIn2 As Boolean
SelectedIn2 = False

For i = 0 To SumDemands - 1
  SelectedIn1 = False
  SelectedIn2 = False
  AnItemHasBeenSelected = False
  Randomize
  r = Rnd

  If (r <= CDF(0) And demand(0) > 0) Then
     ItemsTypeOrdered(i) = itemsType(0)
     ItemsOrdered(i) = items(0)
     ItemsOrdered2(i) = items2(0)
     demand(0) = demand(0) - 1

     AnItemHasBeenSelected = True
     SelectedIn1 = True

   ElseIf (r <= CDF(0) And demand(0) = 0) Then
      For t = 1 To UBound(items) - 1
          If (demand(t) > 0) Then
            ItemsTypeOrdered(i) = itemsType(t)
            ItemsOrdered(i) = items(t)
            ItemsOrdered2(i) = items2(t)
            demand(t) = demand(t) - 1
```



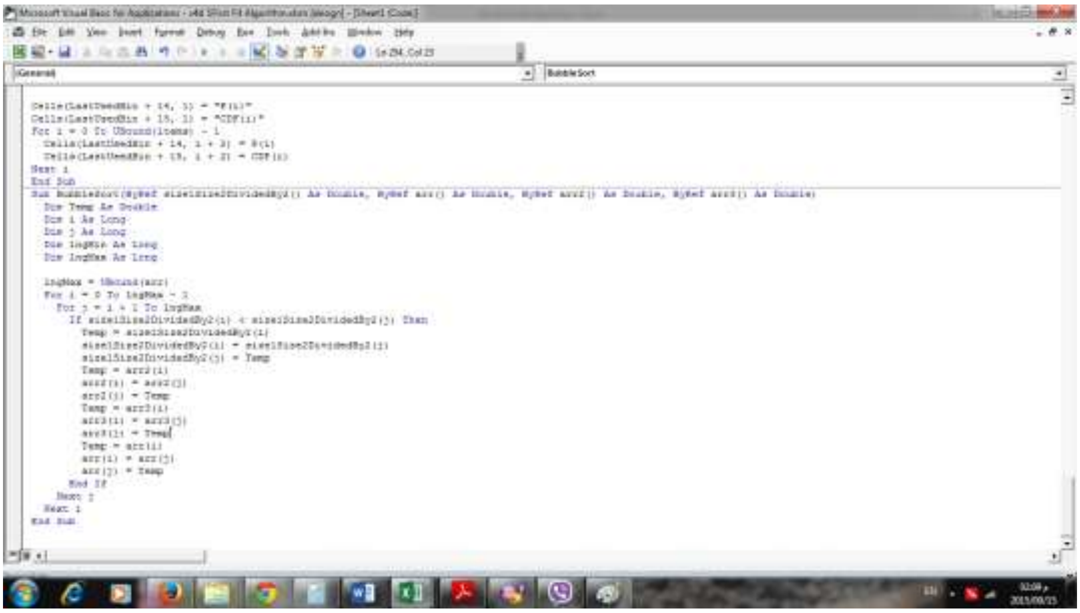
```
  If (AnItemHasBeenSelected = False) Then

  For k = 0 To UBound(items) - 2

     If (r > CDF(k) And r <= CDF(k + 1) And demand(k) > 0 And AnItemHasBeenSelected = False) Then
       ItemsTypeOrdered(i) = itemsType(k)
       ItemsOrdered(i) = items(k)
       ItemsOrdered2(i) = items2(k)
       demand(k) = demand(k) - 1
       AnItemHasBeenSelected = True
       SelectedIn2 = True
       Exit For

     ElseIf (r > CDF(k) And r <= CDF(k + 1) And demand(k) = 0 And AnItemHasBeenSelected = False) Then
      For t = k + 1 To UBound(items) - 1
          If (demand(t) > 0) Then
            ItemsTypeOrdered(i) = itemsType(t)
            ItemsOrdered(i) = items(t)
            ItemsOrdered2(i) = items2(t)
            demand(t) = demand(t) - 1
            AnItemHasBeenSelected = True
            SelectedIn2 = True
            Exit For
          End If
      Next t
      Exit For
     End If
  Next k
  End If
   If (Not SelectedIn1 And Not SelectedIn2) Then
   r = Rnd
   i = i - 1
  End If
```



```
ReDim Bins(SumDemands) As clsBin
For i = 0 To SumDemands
   Set Bins(i) = New clsBin
   Bins(i).updateLastUsedPlace = 1
   Bins(i).updateSize = 0
   Bins(i).updateSize2 = 0
Next i
Dim LastUsedBin As Double
LastUsedBin = 0
For i = 0 To SumDemands - 1
  foundAplaceForTheItem = False
  For j = 0 To LastUsedBin
    If (Bins(j).sizeOfBin + ItemsOrdered(i) <= MaximumSizeOfABin And Bins(j).size2OfBin + ItemsOrdered2(i) <= MaximumSize2OfABin) Then
       Bins(j).updateSize = Bins(j).sizeOfBin + ItemsOrdered(i)
       Bins(j).updateSize2 = Bins(j).size2OfBin + ItemsOrdered2(i)
       Bins(j).updateLastUsedPlace = Bins(j).LastUsedPlaceOfBin + 1
       Cells(j + 5, Bins(j).LastUsedPlaceOfBin + 1) = ItemsTypeOrdered(i)
       foundAplaceForTheItem = True
       Exit For
    End If
  Next j
  If (Not foundAplaceForTheItem) Then
    LastUsedBin = LastUsedBin + 1
    Cells(LastUsedBin + 5, 1) = "Bin " & CStr(LastUsedBin + 1)
    Bins(LastUsedBin).updateSize = Bins(LastUsedBin).sizeOfBin + ItemsOrdered(i)
    Bins(LastUsedBin).updateSize2 = Bins(LastUsedBin).size2OfBin + ItemsOrdered2(i)
    Bins(LastUsedBin).updateLastUsedPlace = Bins(LastUsedBin).LastUsedPlaceOfBin + 1
    Cells(LastUsedBin + 5, Bins(LastUsedBin).LastUsedPlaceOfBin + 1) = ItemsTypeOrdered(i)
    foundAplaceForTheItem = True
  End If
Next i
Dim dTotal As Double
Dim dTotal2 As Double
Dim dAverage As Double
Dim dAverage2 As Double
```

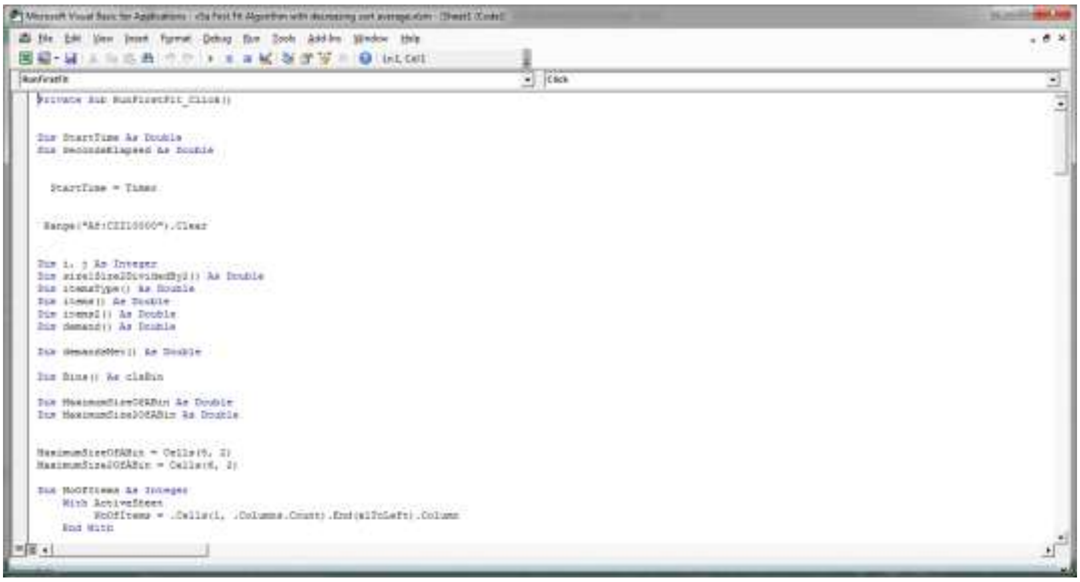


(General)
dTotal = 0
dTotal2 = 0
For j = 0 To LastUsedBin
Next j
dAverage = dTotal / (LastUsedBin + 1)
dAverage2 = dTotal2 / (LastUsedBin + 1)
Cells(LastUsedBin + 9, 1) = "Avg Volume Size"
Cells(LastUsedBin + 9, 2) = "Avg Volume Size2"
Cells(LastUsedBin + 10, 1) = dAverage
Cells(LastUsedBin + 10, 2) = dAverage2
Cells(LastUsedBin + 11, 1) = "Run Time (in Seconds)"
SecondsElapsed = Round(Timer - StartTime, 2)
Cells(LastUsedBin + 11, 2) = SecondsElapsed
Cells(LastUsedBin + 12, 1) = "Size1 after random ordering"
Cells(LastUsedBin + 13, 1) = "Size2 after random ordering"
Cells(LastUsedBin + 12, i + 2) = ItemsOrdered(i)
Cells(LastUsedBin + 13, i + 2) = ItemsOrdered2(i)
Next i

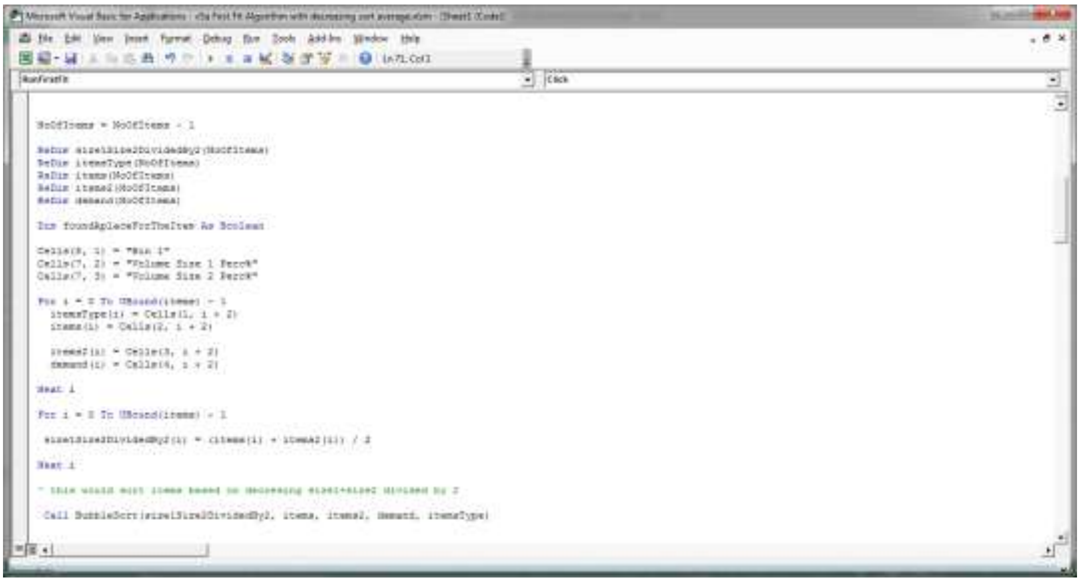


(General)
BubbleSort
Cells(LastUsedBin + 15, 1) = "CDF(i)"
For i = 0 To UBound(items) - 1
Next i
End Sub
Dim Temp As Double
Dim i As Long
Dim j As Long
Dim lngMax As Long
For j = i + 1 To lngMax
sizel1Size2DividedBy2(j) = Temp
Temp = arr2(i)
arr2(j) = Temp
Temp = arr3(i)
arr3(i) = arr3(j)
Temp = arr(i)
arr(i) = arr(j)
arr(j) = Temp
End If
Next j
Next i
End Sub

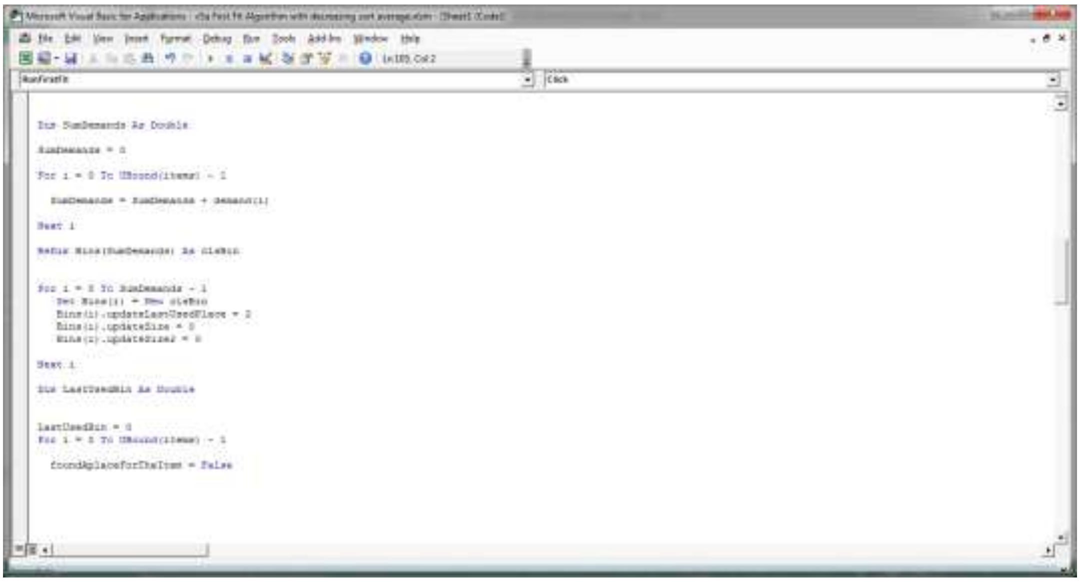
This is the configuration code of the algorithm 9

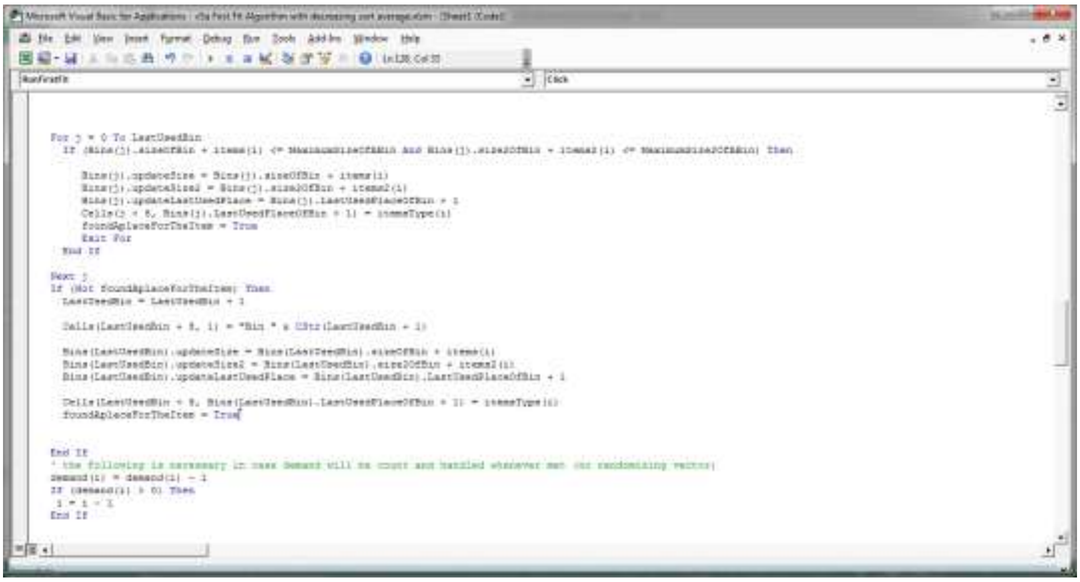


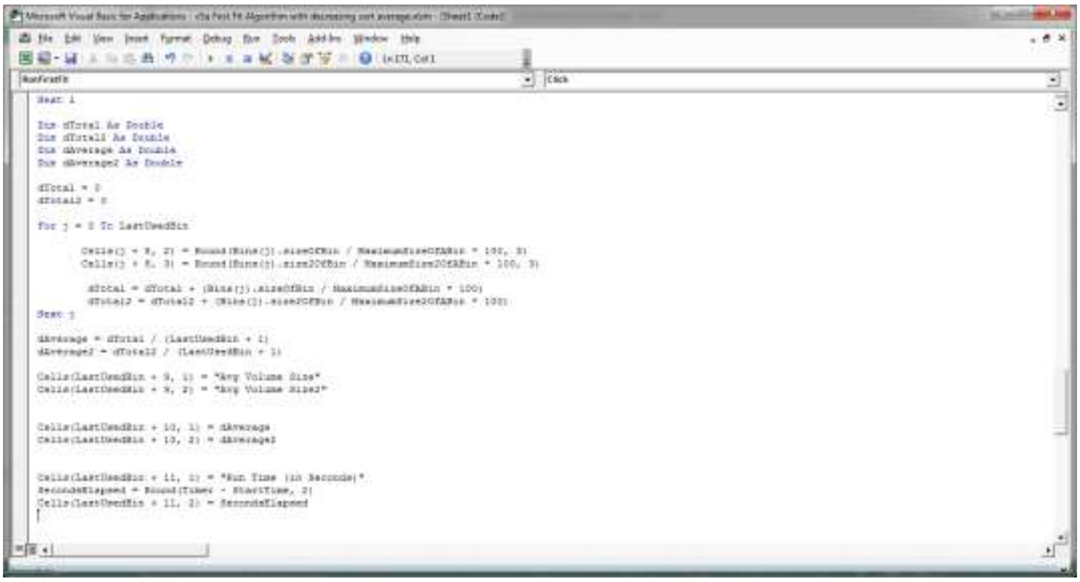
```
Private Sub RunFirstFit_Click()


Dim StartTime As Double
Dim SecondsElapsed As Double


  StartTime = Timer


 Range("A6:CII10000").Clear


Dim i, j As Integer
Dim size1Size2DividedBy2() As Double
Dim itemsType() As Double
Dim items() As Double
Dim items2() As Double
Dim demand() As Double

Dim demandsMet() As Double

Dim Bins() As clsBin

Dim MaximumSizeOfABin As Double
Dim MaximumSize2OfABin As Double


MaximumSizeOfABin = Cells(5, 2)
MaximumSize2OfABin = Cells(6, 2)

Dim NoOfItems As Integer
    With ActiveSheet
        NoOfItems = .Cells(1, .Columns.Count).End(xlToLeft).Column
    End With
```

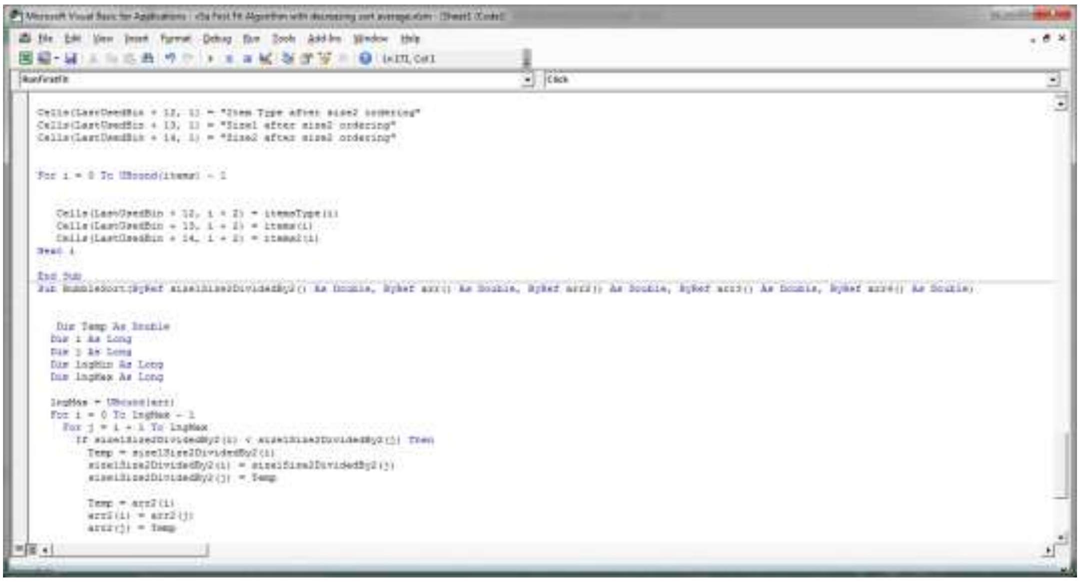


```
NoOfItems = NoOfItems - 1

ReDim size1Size2DividedBy2(NoOfItems)
ReDim itemsType(NoOfItems)
ReDim items(NoOfItems)
ReDim items2(NoOfItems)
ReDim demand(NoOfItems)

Dim foundAplaceForTheItem As Boolean

Cells(8, 1) = "Bin 1"
Cells(7, 2) = "Volume Size 1 Perc%"
Cells(7, 3) = "Volume Size 2 Perc%"

For i = 0 To UBound(items) - 1
  itemsType(i) = Cells(1, i + 2)
  items(i) = Cells(2, i + 2)

  items2(i) = Cells(3, i + 2)
  demand(i) = Cells(4, i + 2)

Next i

For i = 0 To UBound(items) - 1

 size1Size2DividedBy2(i) = (items(i) + items2(i)) / 2

Next i

' this would sort items based on decreasing size1+size2 divided by 2

 Call BubbleSort(size1Size2DividedBy2, items, items2, demand, itemsType)
```

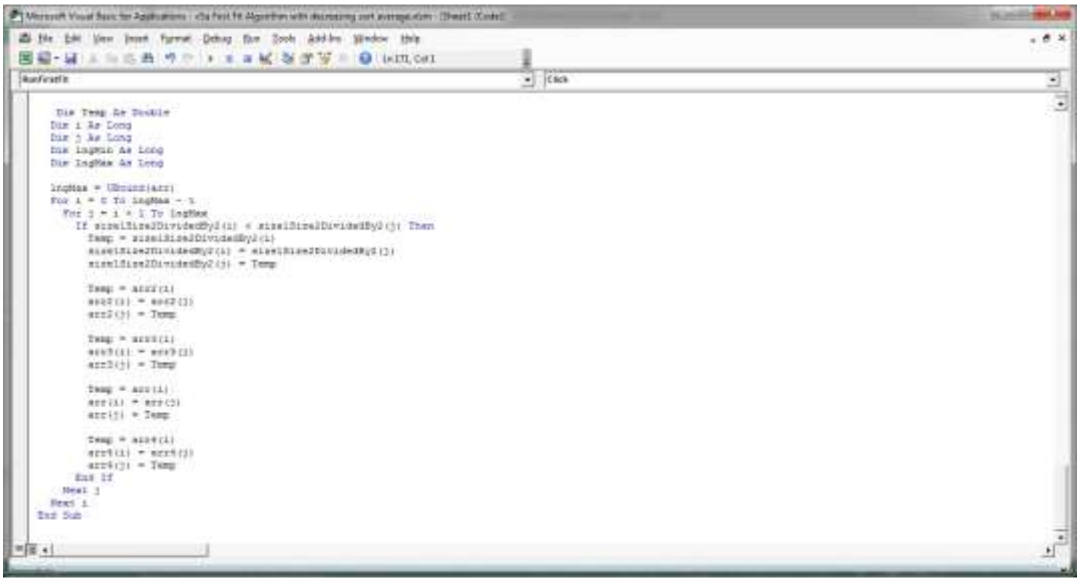

Microsoft Visual Basic for Applications
File Edit View Insert Format Debug Run Tools Add-Ins Window Help
RunFirstFit
Click
Dim SumDemands As Double
SumDemands = 0
For i = 0 To UBound(items) - 1
SumDemands = SumDemands + demand(i)
Next i
ReDim Bins(SumDemands) As clsBin
For i = 0 To SumDemands - 1
Set Bins(i) = New clsBin
Bins(i).updateLastUsedPlace = 2
Bins(i).updateSize = 0
Bins(i).updateSize2 = 0
Next i
Dim LastUsedBin As Double
LastUsedBin = 0
For i = 0 To UBound(items) - 1
foundAplaceforTheItem = False


Microsoft Visual Basic for Applications
File Edit View Insert Format Debug Run Tools Add-Ins Window Help
RunFirstFit
Click
For j = 0 To LastUsedBin
If (Bins(j).sizeOfBin + items(i) <= MaximumSizeOfABin And Bins(j).size2OfBin + items2(i) <= MaximumSize2OfABin) Then
Bins(j).updateSize = Bins(j).sizeOfBin + items(i)
Bins(j).updateSize2 = Bins(j).size2OfBin + items2(i)
Bins(j).updateLastUsedPlace = Bins(j).LastUsedPlaceOfBin + 1
Cells(j + 8, Bins(j).LastUsedPlaceOfBin + 1) = itemsType(i)
foundAplaceForTheItem = True
Exit For
End If
Next j
If (Not foundAplaceForTheItem) Then
LastUsedBin = LastUsedBin + 1
Cells(LastUsedBin + 8, 1) = "Bin " & CStr(LastUsedBin + 1)
Bins(LastUsedBin).updateSize = Bins(LastUsedBin).sizeOfBin + items(i)
Bins(LastUsedBin).updateSize2 = Bins(LastUsedBin).size2OfBin + items2(i)
Bins(LastUsedBin).updateLastUsedPlace = Bins(LastUsedBin).LastUsedPlaceOfBin + 1
Cells(LastUsedBin + 8, Bins(LastUsedBin).LastUsedPlaceOfBin + 1) = itemsType(i)
foundAplaceForTheItem = True
End If
demand(i) = demand(i) - 1
If (demand(i) > 0) Then
i = i - 1
End If


Microsoft Visual Basic for Applications
File Edit View Insert Format Debug Run Tools Add-Ins Window Help
RunFirstFit
Click
Next i
Dim dTotal As Double
Dim dTotal2 As Double
Dim dAverage As Double
Dim dAverage2 As Double
dTotal = 0
dTotal2 = 0
For j = 0 To LastUsedBin
Cells(j + 8, 2) = Round(Bins(j).sizeOfBin / MaximumSizeOfABin * 100, 3)
Cells(j + 8, 3) = Round(Bins(j).size2OfBin / MaximumSize2OfABin * 100, 3)
dTotal = dTotal + (Bins(j).sizeOfBin / MaximumSizeOfABin * 100)
dTotal2 = dTotal2 + (Bins(j).size2OfBin / MaximumSize2OfABin * 100)
Next j
dAverage = dTotal / (LastUsedBin + 1)
dAverage2 = dTotal2 / (LastUsedBin + 1)
Cells(LastUsedBin + 9, 1) = "Avg Volume Size"
Cells(LastUsedBin + 9, 2) = "Avg Volume Size2"
Cells(LastUsedBin + 10, 1) = dAverage
Cells(LastUsedBin + 10, 2) = dAverage2
Cells(LastUsedBin + 11, 1) = "Run Time (in Seconds)"
SecondsElapsed = Round(Timer - StartTime, 2)
Cells(LastUsedBin + 11, 2) = SecondsElapsed

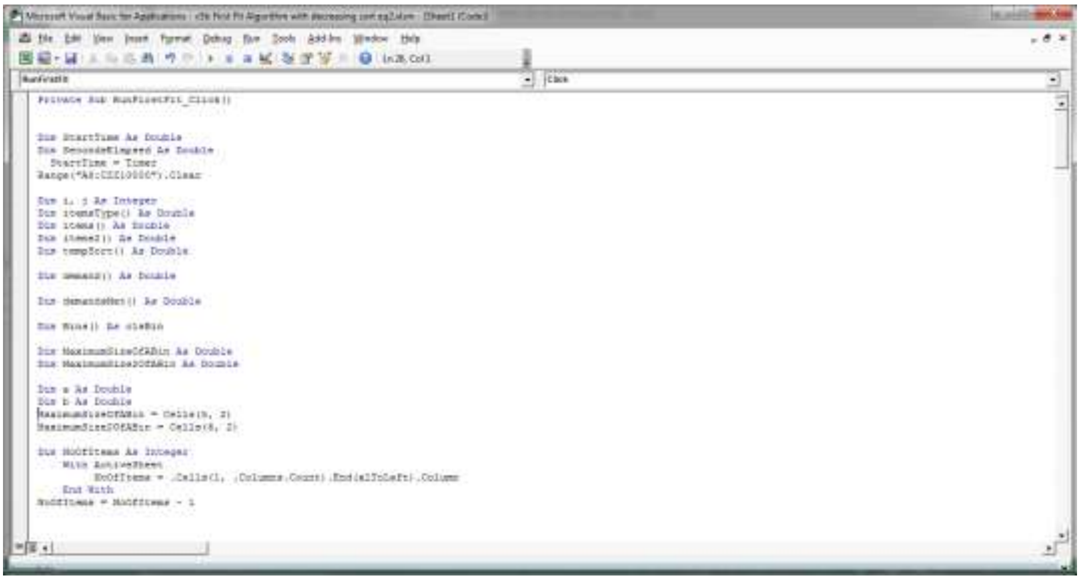


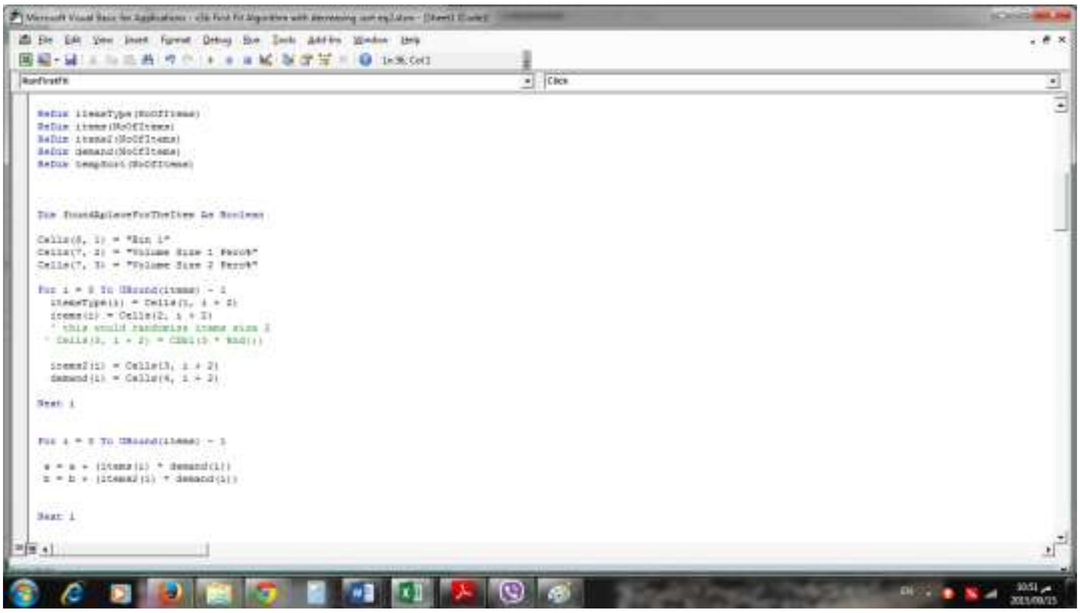
```
Cells(LastUsedBin + 12, 1) = "Item Type after size2 ordering"
Cells(LastUsedBin + 13, 1) = "Size1 after size2 ordering"
Cells(LastUsedBin + 14, 1) = "Size2 after size2 ordering"


For i = 0 To UBound(items) - 1


   Cells(LastUsedBin + 12, i + 2) = itemsType(i)
   Cells(LastUsedBin + 13, i + 2) = items(i)
   Cells(LastUsedBin + 14, i + 2) = items2(i)
Next i

End Sub
Sub BubbleSort(ByRef size1Size2DividedBy2() As Double, ByRef arr() As Double, ByRef arr2() As Double, ByRef arr3() As Double, ByRef arr4() As Double)


   Dim Temp As Double
  Dim i As Long
  Dim j As Long
  Dim lngMin As Long
  Dim lngMax As Long

  lngMax = UBound(arr)
  For i = 0 To lngMax - 1
    For j = i + 1 To lngMax
      If size1Size2DividedBy2(i) < size1Size2DividedBy2(j) Then
        Temp = size1Size2DividedBy2(i)
        size1Size2DividedBy2(i) = size1Size2DividedBy2(j)
        size1Size2DividedBy2(j) = Temp

        Temp = arr2(i)
        arr2(i) = arr2(j)
        arr2(j) = Temp
```



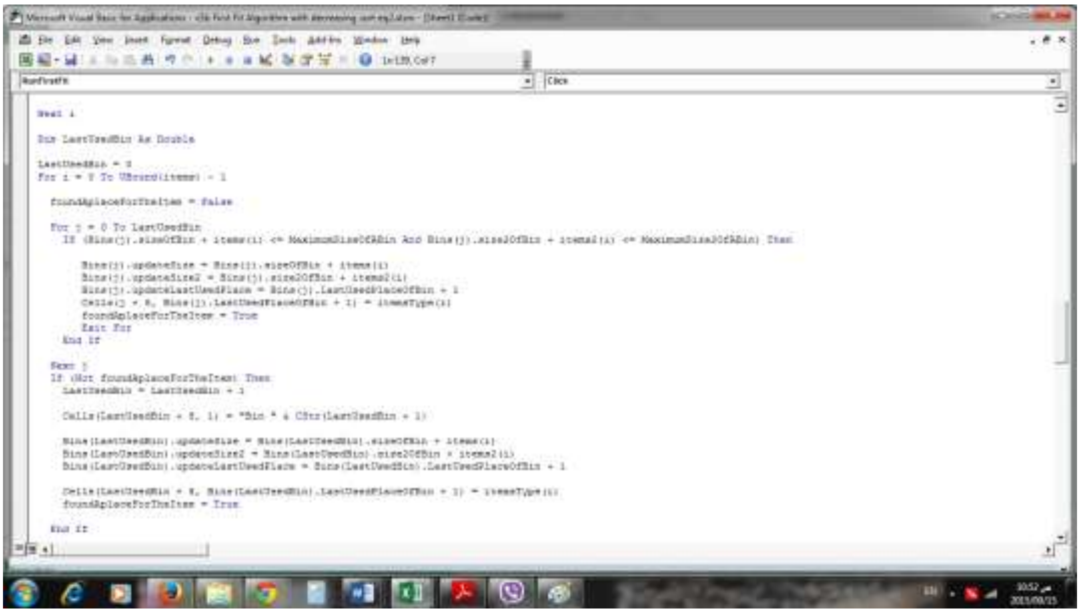
```
   Dim Temp As Double
  Dim i As Long
  Dim j As Long
  Dim lngMin As Long
  Dim lngMax As Long

  lngMax = UBound(arr)
  For i = 0 To lngMax - 1
    For j = i + 1 To lngMax
      If size1Size2DividedBy2(i) < size1Size2DividedBy2(j) Then
        Temp = size1Size2DividedBy2(i)
        size1Size2DividedBy2(i) = size1Size2DividedBy2(j)
        size1Size2DividedBy2(j) = Temp

        Temp = arr2(i)
        arr2(i) = arr2(j)
        arr2(j) = Temp

        Temp = arr3(i)
        arr3(i) = arr3(j)
        arr3(j) = Temp

        Temp = arr(i)
        arr(i) = arr(j)
        arr(j) = Temp

        Temp = arr4(i)
        arr4(i) = arr4(j)
        arr4(j) = Temp
      End If
    Next j
  Next i
End Sub
```

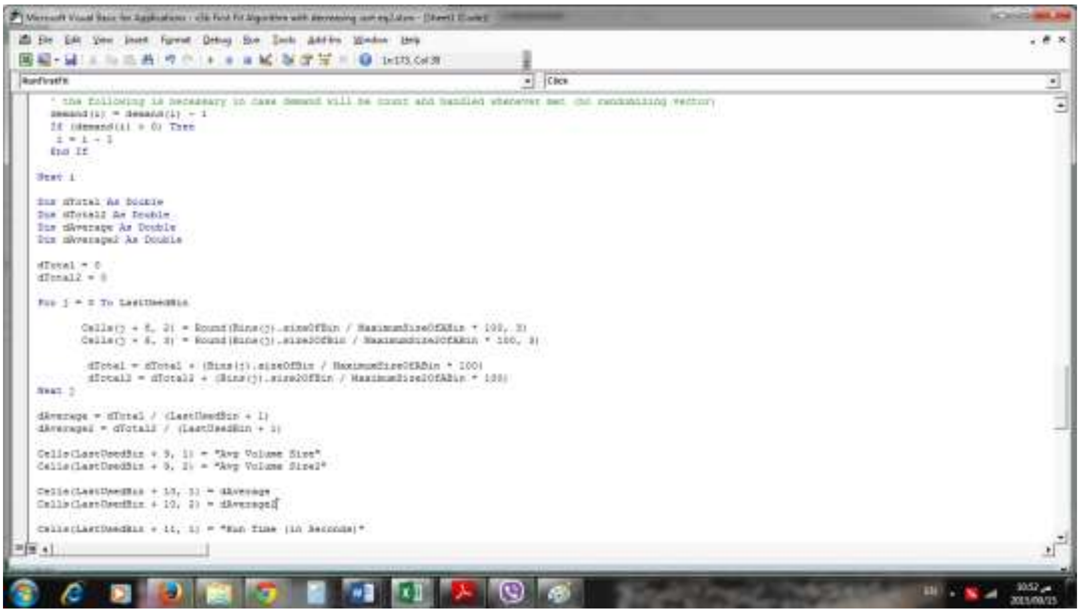
This is the configuration code of the algorithm 10

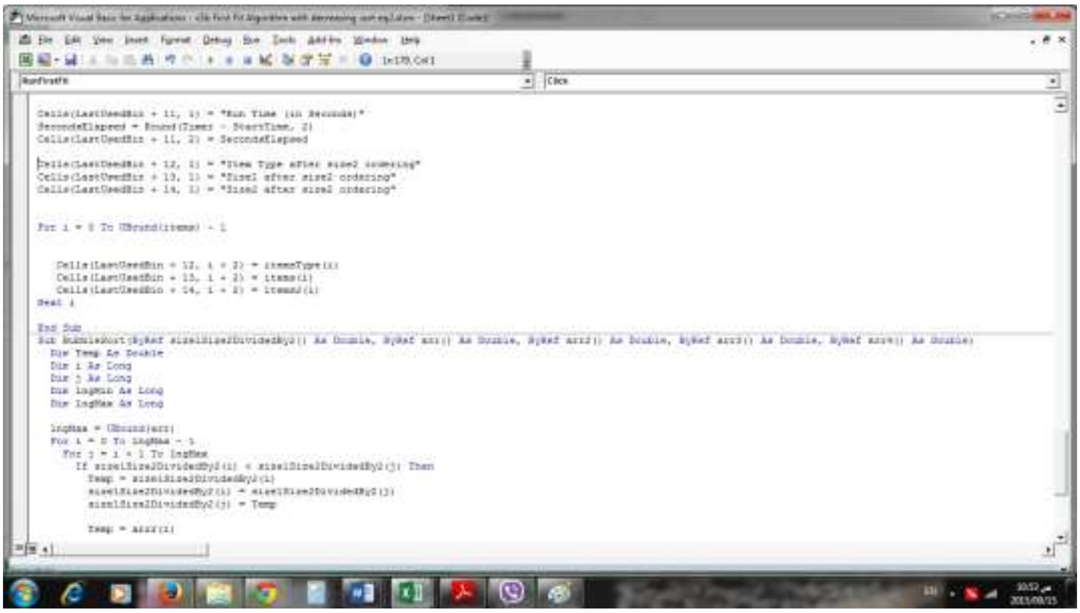




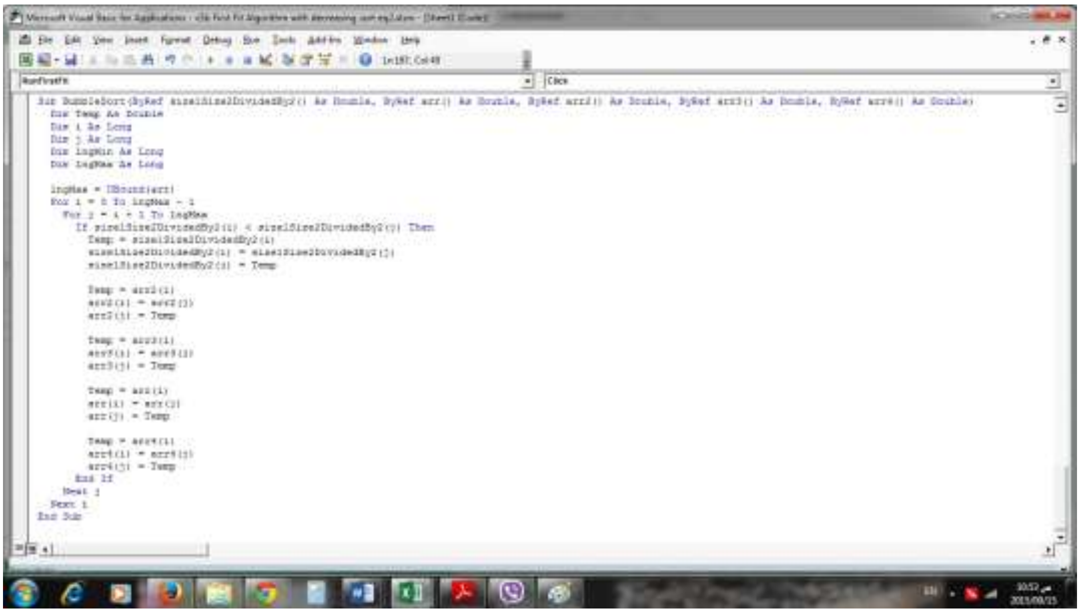

File Edit View Insert Format Debug Run Tools Add-Ins Window Help
a = a / MaximumSizeOfABin
b = b / MaximumSizeOfABin
Cells(5, 3) = "a"
Cells(5, 4) = "b"
Cells(6, 3) = a
Cells(6, 4) = b
tempSort(i) = (a * items(i) + b * items2(i)) / 2
Call BubbleSort(tempSort, items, items2, demand, itemsType)
Dim SumDemands As Double
SumDemands = 0
SumDemands = SumDemands + demand(i)
Next i
Set Bins(i) = New clsBin
Bins(i).updateSize = 0
Bins(i).updateSize2 = 0


File Edit View Insert Format Debug Run Tools Add-Ins Window Help
Dim LastUsedBin As Double
foundAplaceForTheItem = False
For j = 0 To LastUsedBin
foundAplaceForTheItem = True
Exit For
End If
If (Not foundAplaceForTheItem) Then
LastUsedBin = LastUsedBin + 1
foundAplaceForTheItem = True
End If


File Edit View Insert Format Debug Run Tools Add-Ins Window Help
Dim dTotal As Double
Dim dTotal2 As Double
Dim dAverage As Double
Dim dAverage2 As Double
dTotal = 0
dTotal2 = 0
For j = 0 To LastUsedBin
dAverage = dTotal / (LastUsedBin + 1)
dAverage2 = dTotal2 / (LastUsedBin + 1)
Cells(LastUsedBin + 9, 1) = "Avg Volume Size"
Cells(LastUsedBin + 9, 2) = "Avg Volume Size2"
Cells(LastUsedBin + 10, 1) = dAverage
Cells(LastUsedBin + 10, 2) = dAverage2
Cells(LastUsedBin + 11, 1) = "Run Time (in Seconds)"



```
Cells(LastUsedBin + 11, 1) = "Run Time (in Seconds)"
SecondsElapsed = Round(Timer - StartTime, 2)
Cells(LastUsedBin + 11, 2) = SecondsElapsed

Cells(LastUsedBin + 12, 1) = "Item Type after size2 ordering"
Cells(LastUsedBin + 13, 1) = "Size1 after size2 ordering"
Cells(LastUsedBin + 14, 1) = "Size2 after size2 ordering"


For i = 0 To UBound(items) - 1


   Cells(LastUsedBin + 12, i + 2) = itemsType(i)
   Cells(LastUsedBin + 13, i + 2) = items(i)
   Cells(LastUsedBin + 14, i + 2) = items2(i)
Next i

End Sub
Sub BubbleSort(ByRef size1Size2DividedBy2() As Double, ByRef arr() As Double, ByRef arr2() As Double, ByRef arr3() As Double, ByRef arr4() As Double)
  Dim Temp As Double
  Dim i As Long
  Dim j As Long
  Dim lngMin As Long
  Dim lngMax As Long

  lngMax = UBound(arr)
  For i = 0 To lngMax - 1
    For j = i + 1 To lngMax
      If size1Size2DividedBy2(i) < size1Size2DividedBy2(j) Then
        Temp = size1Size2DividedBy2(i)
        size1Size2DividedBy2(i) = size1Size2DividedBy2(j)
        size1Size2DividedBy2(j) = Temp

        Temp = arr2(i)
```



```
Sub BubbleSort(ByRef size1Size2DividedBy2() As Double, ByRef arr() As Double, ByRef arr2() As Double, ByRef arr3() As Double, ByRef arr4() As Double)
  Dim Temp As Double
  Dim i As Long
  Dim j As Long
  Dim lngMin As Long
  Dim lngMax As Long

  lngMax = UBound(arr)
  For i = 0 To lngMax - 1
    For j = i + 1 To lngMax
      If size1Size2DividedBy2(i) < size1Size2DividedBy2(j) Then
        Temp = size1Size2DividedBy2(i)
        size1Size2DividedBy2(i) = size1Size2DividedBy2(j)
        size1Size2DividedBy2(j) = Temp

        Temp = arr2(i)
        arr2(i) = arr2(j)
        arr2(j) = Temp

        Temp = arr3(i)
        arr3(i) = arr3(j)
        arr3(j) = Temp

        Temp = arr(i)
        arr(i) = arr(j)
        arr(j) = Temp

        Temp = arr4(i)
        arr4(i) = arr4(j)
        arr4(j) = Temp
      End If
    Next j
  Next i
End Sub
```